Tampere University

Sini Tiistola

REINFORCEMENT Q-LEARNING FOR MODEL-FREE OPTIMAL CONTROL

Real-time implementation and challenges

Faculty of Engineering and
Natural Sciences
Master of Science Thesis
August 2019

ABSTRACT

Sini Tiistola: Reinforcement Q-learning for model-free optimal control: Real-time implementation and challenges
Master of Science Thesis
Tampere University
Automation Engineering
August 2019

Traditional feedback control methods are often model-based and the mathematical system models need to be identified before or during control. A reinforcement learning method called Q-learning can be used for model-free state feedback control. In theory, the optimal adaptive control is learned online without a system model with Q-learning. This data-driven learning is based on the output or state measurements and input control actions of the system. Theoretical results are promising, but the real-time applications are not widely used. Real-time implementation can be difficult because of e.g. hardware restrictions and stability issues.

This research aimed to determine whether a set of already existing Q-learning algorithms is capable of learning the optimal control in real-time applications. Both batch offline and adaptive online algorithms were chosen for this study. The selected Q-learning algorithms were implemented in a marginally stable linear system and an unstable nonlinear system using the Quanser QUBE™-Servo 2 experiment with an inertia disk and an inverted pendulum attachments. The results learned from the real-time system were compared to the theoretical Q-learning results when a simulated system model was used.

The results proved that the chosen algorithms solve the Linear Quadratic Regulator (LQR) problem with the theoretical linear system model. The algorithm chosen for the nonlinear system approximated the Hamilton-Jacobi-Bellman equation solution with the theoretical model, when the inverted pendulum was balanced upright. The results also showed that some challenges caused by the real-time system can be avoided by a proper selection of control noise. These include e.g. constrained input voltage and measurement disturbances such as small measurement noise and quantization due to measurement resolution. In the best, but rare, adaptive test cases, a near optimal policy was learned online for the linear real-time system. However, learning is reliable only with some batch learning methods. Lastly, some suggestions for improvements were proposed for Q-learning to be more suitable for real-time applications.

Keywords: Actor-Critic, Feedback Control, Least Squares, Linear Quadratic Regulator, Model-free Control, Neural Networks, Optimal Control, Policy Iteration, Q-learning, Reinforcement learning, Stochastic Gradient Descent, Value Iteration

TIIVISTELMÄ

Sini Tiistola: Q-vahvistusoppiminen mallittomassa optimisäädössä: Reaaliaikainen säätö ja haasteet
Diplomityö
Tampereen yliopisto
Automaatiotekniikan tutkinto-ohjelma
Elokuu 2019

Perinteiset takaisinkytketyt säätömetodit ovat usein mallipohjaisia ja matemaattiset systeemimallit tulee identifioida ennen säätöä tai sen aikana. Erästä vahvistusoppimisen menetelmää, Q-oppimista (Q-learning), voidaan käyttää mallittomaan tilasäätöön. Teoriassa adaptiivinen optimisäätö opitaan ilman systeemimallia Q-oppimisella. Tämä tietopohjainen oppiminen perustuu vain systeemin ulostulo- tai tilamittauksiin ja sisään meneviin ohjauksiin. Teoreettiset tulokset ovat lupaavia, mutta reaaliaikaiset sovellutukset eivät ole laajalti käytettyjä. Reaaliaikainen toteutus voi olla hankalaa esimerkiksi toimilaitteiden rajoitteiden sekä stabiiliusongelmien vuoksi.

Tämän tutkimuksen tavoitteena oli selvittää, voiko joukko jo olemassa olevia Q-oppimisalgoritmeja oppia optimisäädön reaaliaikaisissa sovellutuksissa. Tutkimukseen valittiin adaptiivisia online-säätömenetelmiä, sekä offline-menetelmiä. Valitut Q-oppimisalgoritmit toteutettiin marginaalisesti stabiilissa lineaarisessa systeemissä sekä epästabiilissa epälineaarisessa systeemissä käyttäen Quanser QUBE™-Servo 2 laitetta inertialevyn tai kääntöheilurin kanssa. Oikealla laitteella opittuja tuloksia verrattiin simuloidulla mallilla saatuihin tuloksiin.

Tulokset osoittivat valittujen algoritmien ratkaisevan LQR (Linear Quadratic Regulator) -ongelman valitulle lineaariselle järjestelmälle teoreettisella mallilla. Epälineaariselle systeemille valittu algoritmi approksimoi HJB (Hamilton-Jacobi-Bellman) -yhtälön ratkaisun teoreettisella mallilla, kun kääntöheiluria tasapainotettiin pystyasennossa. Tulokset näyttivät myös, että sopivalla ohjauskohinan valinnalla voidaan välttää joitakin oikean laitteen aiheuttamia haasteita. Näitä ovat muun muassa rajoitettu ohjausjännite sekä mittaushäiriöt, kuten pieni mittauskohina ja mittausresoluution vuoksi kvantisoituneet mittaukset. Parhaimmissa, mutta harvoissa, adaptiivisissa testitapauksissa opittiin lähes optimaalinen säätöpolitiikka reaaliaikaisesti oikealla laitteella. Kuitenkin, oppiminen oli luotettavaa vain joillakin offline-oppimismenetelmillä. Lopulta, joitakin parannusehdotuksia esitettiin, jotta Q-oppiminen soveltuisi paremmin reaaliaikaisiin sovellutuksiin.

Avainsanat: LQR, neuroverkot, malliton säätö, optimisäätö, Q-oppiminen, säätötekniikka, takaisinkytketty säätö, tekoäly, vahvistusoppiminen

PREFACE

This thesis was a part of the MIDAS project at Tampere University. I would like to thank my supervisors, Professors Matti Vilkko and Risto Ritala, for the help and advice throughout my thesis process, but also for the freedom.

Tampere, 16 August 2019

Sini Tiistola

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ADHDP	Action Dependent Heuristic Dynamic Programming
ADP	Adaptive Dynamic Programming
ARE	Algebraic Riccati Equation
HJB	Hamilton-Jacobi-Bellman equation
LQR	Linear Quadratic Regulator
LS	Least Squares with weight matrix value update
LS2	Least Squares with kernel matrix value update
NN	Neural network
PI	Policy Iteration
RLS	Recursive Least Squares
SGD	Stochastic Gradient Descent
VI	Value Iteration
PE	Persistence of Excitation
Q-function	Quality function

$\mathbb{R}$	Real numbers
$(\cdot)^*$	Optimal solution
$\varepsilon_i, \varepsilon_j, \varepsilon_m$	Value update and Q-function and model network convergence limits
ϵ_k	Discrete-time control noise at time k
$\alpha_{sgd}, \alpha_a, \alpha_c, \alpha_m$	Actor, critic and model network learning rates
β, θ	Pendulum link and rotary angle of Quanser QUBE™-Servo 2
γ	User-defined discount factor
δ	Recursive least squares covariance initializing parameter
λ	Recursive least squares discounting factor
μ	Data vector
$\sigma(\cdot)$	Activation function $(\tanh)$
φ	Regression vector
Φ	Least squares matrix of regression vectors
$\phi(\cdot)$	Basis function
Ψ_k	Least squares data matrix
A	Discrete-time linear system model matrix
a	Recursive least squares weighting parameter
B	Discrete-time linear system input matrix
C	Discrete-time linear system output matrix
$c(x_k)$	Discrete-time nonlinear system output matrix
D	Discrete-time linear system feedthrough matrix
E_a, E_c, E_m, E_{nw}	Actor, critic, model and general neural network mean square error
e	Temporal difference error
e_a, e_c, e_m	Actor, critic and model network estimation error
$f(x_k)$	Discrete-time nonlinear system inner state dynamics
$g(x_k)$	Discrete-time nonlinear system input coefficient matrix
$h(x_k)$	Control policy at iteration step j
i, j, k	Iteration, iteration step and time index
K	Control gain vector
L	Recursive least squares update matrix
M	Batch size
N	Number of data samples
n	Observability index
$n_u, n_x, n_y, n_z, n_{\bar{z}}$	Number of inputs, states, outputs and elements in z_k and $\bar{z}_k$
n_{sets}, n_{nw}	Number of learning episodes and network neurons

P	Recursive least squares covariance matrix
Q, Q_y	LQR state and output weighting matrices
$Q_h(x_k, u_k)$	Q-function with control policy $h(x_k)$
R	LQR control weighting matrix
$r(x_k, u_k)$	One-step cost
S, s	Symmetric kernel matrix and its element for full state measurements
$S_{uu}, S_{ux},$	Lower matrix elements of the matrix S
S_{xx}, S_{xu}	Upper matrix elements of the matrix S
T	Symmetric kernel matrix for output measurements
T_n	Toeplitz matrix
$T_{uu}, T_{u\bar{u}}, T_{u\bar{y}}$	Lower matrix elements of the matrix T
$T_{\bar{u}u}, T_{\bar{u}\bar{u}}, T_{\bar{u}\bar{y}}$	Upper matrix elements of the matrix T
$T_{\bar{y}u}, T_{\bar{y}\bar{u}}, T_{\bar{y}\bar{y}}$	Middle matrix elements of the matrix T
T_0	Initial value of the matrix T
U_n	Controllability matrix
u_k	Discrete-time control at time k
$\bar{u}_k$	Vector of previous controls
$V_h(x_k)$	Quadratic cost function and value function
V_n	Observability matrix
v_a, v_c, v_m	Actor, critic and model network hidden layer weight matrices
W	Vector of upper triangular terms of matrix S
W_a, W_c, W_m	Actor, critic and model network weight matrix
X	Solution of Algebraic Riccati equation
Y	Least squares data matrix
y_k	Output measurement at time k
$\bar{y}_k$	Vector of previous measurements
x_k	Discrete-time state at time k
$\bar{x}_k$	Discrete-time state replacement at time k
z_k	Vector of states and measurements at time k
$\bar{z}_k$	Vector of replacement states and measurements at time k

1. INTRODUCTION

Traditional feedback control methods often need mathematical system models and require model identification [6]. Reinforcement Q-learning provides tools for model-free optimal adaptive control without system identification [16][36]. It learns the optimal state feedback control online or offline only by collecting output measurements and control inputs.

Numerous different discrete-time and continuous-time Q-learning methods for linear and nonlinear systems are already implemented in literature [5][14]-[17][19][20][22][26] [27][31]-[34][36][42][44]. In theory, Q-learning converges to the optimal control solution, but most of the research uses only simulated models. Recent studies on Q-learning and other adaptive dynamic programming reinforcement learning methods have started to include more focus on real-time applications such as constrained control in [33][34] and systems with disturbances in [38]. However, only few papers, e.g. [8][31], use real data instead of simulated data and apply Q-learning offline.

Q-learning is not widely used in real-time systems yet and therefore the aim of this study is to implement some of the already existing Q-learning methods and to analyse their performance in real-time applications and to try to find solutions to the possible challenges and threats opposed by the real-time environment. According to literature [3][16][23][41], real-time implementation can be problematic due to hardware restrictions, stability issues and slow convergence of the traditional iterative Q-learning methods.

1.1 The focus of this thesis

This study focuses on discrete-time model-free Q-learning algorithms. These algorithms learn the optimal state-feedback control with data from a theoretical model. A set of already existing Q-learning algorithms is implemented in linear and nonlinear real-time systems. These algorithms are chosen and modified from the research papers in [2][13][15]-[17][19][23][25][36][42].

In the linear environment, model-free optimal control is implemented using the more traditional iterative Q-learning algorithms, policy iteration (PI) and value iteration (VI). These algorithms solve the Linear Quadratic Regulator (LQR) problem without a system model.

Policy and value iteration are implemented using four different methods from articles [15]-[17][25][36][42]. These methods are two different least squares methods (LS and LS2), recursive least squares (RLS) and stochastic gradient descent (SGD). Similarly, nonlinear model-free control problem is solved using interleaved Q-learning method based on the algorithms in articles [2][13][19][23]. This algorithm is a modification of the policy and value iteration algorithms.

The relations of the algorithms studied in this thesis are shown in Figure 1 using different colours. The grey sections are not used in this thesis. The chosen algorithms are divided into on-policy and off-policy algorithms. Literature [19][21] defines terms on-policy and off-policy in the Q-learning context. On-policy means that the updated policy is also used for control, whereas off-policy methods can update different policy than what is used in the system.

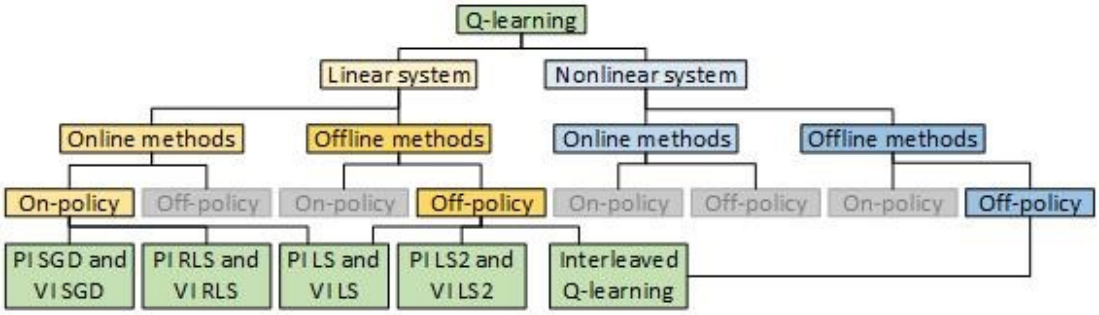

Figure 1. *Q-learning algorithms covered in this thesis*

The real-time system in this thesis is a Quanser QUBE™-Servo 2 environment with an inertia disk and an inverted pendulum attachment. The first one is assumed as a linear system and the latter as a nonlinear system. In reality the input voltage is constrained, full states are not measurable, and the measurements are quantized due to the encoder resolution [1][30]. Theoretical algorithms do not consider these features nor disturbances, which might cause problems.

1.2 The structure of this thesis

Chapter 2 presents the theoretical background of Q-learning in feedback control and the algorithms given in Figure 1 are explained in more detail for systems with measurable full states and systems with only partially measurable states. Chapter 3 focuses on implementation and discusses the challenges and issues when moving from simulated Quanser QUBE™-Servo 2 environment to the real-time environment. Chapters 4 and 5 present the Q-learning results and compare the simulated results with the real-time results. The results are then analysed and conclusions are given in Chapter 5.

2. REINFORCEMENT Q-LEARNING IN FEEDBACK CONTROL

Reinforcement learning is a machine learning method for learning actions, or policies, by observing responses given by the system when performing these actions [16][17]. Reinforcement learning methods called adaptive dynamic programming (ADP) are forward-in-time methods for solving Hamilton-Jacobi-Bellman (HJB) equation [15][16][42][43]. They are used for data-driven adaptive optimal control without system model.

Two common ADP-based reinforcement learning methods are introduced in literature. The first method is called value function approximation (VFA) in [2][4][10][11] [13][21][23]-[25][38]-[41][43] and the other action dependent heuristic dynamic programming (ADHDP) or Q-learning in [5][8][14]-[17][19][20][22][26][27][31]-[34][36][42][44]. The first one, VFA, uses state measurements to calculate the optimal state-feedback control and needs some knowledge of the system. The latter, Q-learning, is a model-free reinforcement learning method. It allows the system to learn the optimal control policy based on only the control actions and their measured responses. Model-free data-driven control therefore removes the need for any system model or identification. Q-learning aims to learn an optimal Quality function (Q-function) and an optimal action or control policy based on the optimal Q-function The Q-function includes all the data of the states and control actions and the optimal control is the control that minimizes the optimal Q-function.

Traditionally, Q-learning algorithms are iterative policy and value iteration algorithms as in [16][17][25][43]. These algorithms iterate the Q-function and control policy so that one is kept constant while the other one is learned and vice versa. While iterative Q-learning methods work in simulated environments, many articles [13][19][23] propose optional methods for nonlinear control called interleaved Q-learning, where both the Q-function and the control policy are learned simultaneously. They also claim that these methods are safer and easier to implement in real-time applications than the iterative methods.

Policy and value iteration can be implemented either by on-policy or off-policy methods. On-policy Q-learning methods learn the optimal control policy while running the system forward in time [13][16][17][19]. One of the common architectures for Q-learning based on-policy algorithms is called in literature [10][16][25][42] the actor-critic structure (Figure 2). The critic updates the Q-function and the actor updates the control policy based on the critic's output.

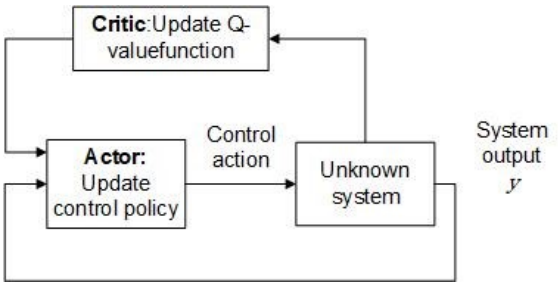

Figure 2. *Actor-Critic structure (modified from [16]-[17])*

In contrast, off-policy Q-learning methods use a behaviour policy to run the system. They collect a set of data and reuse it in the learning phase to learn the optimal control as in [15][19]-[21][33][34][36].

2.1 Reinforcement Q-learning in linear model-free control

Generally, linear discrete-time systems are expressed in time invariant state space form as in [6] as

$$\begin{cases} x_{k+1} = Ax_k + Bu_k \\ u_k = Kx_k + \epsilon_k \\ y_k = Cx_k + Du_k \end{cases} \tag{2.1}$$

where $x_k \in \mathbb{R}^{n_x}$ is the state at time k and n_x is number of states, $u_k \in \mathbb{R}^{n_u}$ is the control at time k and n_u is number of inputs, $y_k \in \mathbb{R}^{n_y}$ is the control at time k and n_y is the number of outputs, $A \in \mathbb{R}^{n_x \times n_x}$ is the state matrix, $B \in \mathbb{R}^{n_x \times n_u}$ is the input matrix, $C \in \mathbb{R}^{n_y \times n_x}$ is the output matrix, $D \in \mathbb{R}^{n_y \times n_u}$ is the feedthrough matrix and ϵ_k the control noise.

Discrete-time Linear Quadratic Regulator (LQR) solves an optimal gain for the system in (2.1). The optimal gain minimizes a quadratic cost function. According to [6][15]-[17][36], the quadratic cost function to be minimized is expressed as

$$V_h(x_k) = \sum_{i_k=k}^{\infty} \gamma^{i_k - k} r\left(x_{i_k}, u_{i_k}\right) , \tag{2.2}$$

where γ is a discounting factor, and $r\left(x_{i_k}, u_{i_k}\right)$ is a one-step cost given as

$$r\left(x_{i_k}, u_{i_k}\right) = {x_{i_k}}^T Q x_{i_k} + {u_{i_k}}^T R u_{i_k}, \tag{2.3}$$

and $Q \in \mathbb{R}^{nxn}$ is a user-defined state weighting matrix and $R \in \mathbb{R}^{mxm}$ is a user-defined control weighting matrix.

The optimal gain that minimizes the quadratic cost function of equation (2.2) with a discounting factor $\gamma = 1$ is given in literature [6][15]-[17][33] as

$$K^* = (R + B^T XB)^{-1} B^T XA \tag{2.4}$$

where X is the discrete-time Algebraic Riccati equation (ARE) solution. The discrete-time ARE for system (2.1) is derived as

$$X = A^T XA - A^T XB(R + B^T XB)^{-1} B^T XA + Q. \tag{2.5}$$

As can be seen, solving the Riccati equation (2.5) needs knowledge of the full dynamics of the system. However, Q-learning is proven to solve this problem without the system model using only knowledge of the states x_k or outputs y_k and control actions u_k in [16][17][33][34][36].

2.1.1 Q-learning with full state measurements

Deriving model-free solution to Riccati equation (2.5) starts in [15][16][17][36] from expressing the quadratic cost function (2.2) as a value function. The quadratic cost function (2.2) is expressed as

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}), V_h(0) = 0 \tag{2.6}$$

where $V_h(x_k)$ is the value function, $r\big(x_k, h(x_k)\big)$ is the one-step cost given in (2.3) at index $i_k = k$ with policy $h(x_k) = u_k$ and γ is a discounting factor. This is also called a Bellman equation.

The optimal value and policy are then given as

$$\begin{cases} V^*(x_k) = \min\limits_{h(\cdot)} \big(r(x_k, h(x_k)) + \gamma V^*(x_{k+1})\big) \\ h^*(x_k) = \arg\min\limits_{u_k} \Big(r\big(x_k, h(x_k)\big) + \gamma V^*(x_{k+1})\Big) \end{cases} \tag{2.7}$$

Using the value function, the optimal Q-function is derived in literature [14]-[17] as

$$Q^*\big(x_k, h(x_k)\big) = r\big(x_k, h(x_k)\big) + \gamma V^*(x_{k+1}). \tag{2.8}$$

With this information equation (2.7) becomes

$$\begin{cases} V^*(x_k) = \min\limits_{h(\cdot)} \Big(Q^*\big(x_k, h(x_k)\big)\Big) \\ h^*(x_k) = \arg\min\limits_{u_k} \Big(Q^*\big(x_k, h(x_k)\big)\Big) \end{cases} \tag{2.9}$$

The general Q-learning Bellman equation is derived by denoting

$$Q_h\big(x_k, h(x_k)\big) = V_h(x_k)\,. \tag{2.10}$$

Combining (2.6) and (2.10) yields

$$Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h\big(x_{k+1}, h(x_{k+1})\big). \tag{2.11}$$

For LQR, the Q-learning Bellman equation is derived in literature [16][17][33][34][36] in the form

$$Q_h(x_k, u_k) = {z_k}^T S z_k = r(x_k, u_k) + \gamma {z_{k+1}}^T S z_{k+1}, \quad (2.12)$$

where $z_k \in \mathbb{R}^{n_z}$, $n_z = n_u + n_x$ is

$$z_k = \begin{bmatrix} x_k \\ h(x_k) \end{bmatrix}, \quad (2.13)$$

$h(x_k) = u_k$ and z_{k+1} is calculated using (2.13) . The symmetric positive definite quadratic kernel matrix S is derived as

$$S = \begin{bmatrix} A^T X A + Q & A^T X B \\ B^T X A & B^T X B + R \end{bmatrix} = \begin{bmatrix} S_{xx} & S_{xu} \\ S_{ux} & S_{uu} \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1l} \\ s_{21} & s_{22} & & s_{2l} \\ \vdots & & \ddots & \vdots \\ s_{l1} & s_{l2} & \cdots & s_{ll} \end{bmatrix}, \quad (2.14)$$

where X is the Riccati equation (2.5) solution, $S \in \mathbb{R}^{n_z x n_z}$ and $S_{xx} \in \mathbb{R}^{n_x x n_x}$, $S_{xu} = {S_{ux}}^T \in \mathbb{R}^{n_x x n_u}$, $S_{uu} \in \mathbb{R}^{n_u x n_u}$ and s are elements of S. The matrix S is learned without a system model and the Riccati equation solution X.

LQR Bellman equation (2.12) in linear approximation form is derived in [16][17] as

$$W^T \phi(z_k) = r(x_k, u_k) + \gamma W^T \phi(z_{k+1}), \quad (2.15)$$

where $W \in \mathbb{R}^{(n_z(n_z+1)/2) x 1}$ is a vector of upper triangular terms of S matrix

$$W = \left[s_{11}, 2s_{12}, \cdots, 2s_{1n_z}, s_{22}, \cdots, 2s_{2n_z}, s_{33}, \cdots, 2s_{3n_z}, \cdots, s_{n_z n_z}\right]^T, \quad (2.16)$$

and $\phi(z_k) \in \mathbb{R}^{(n_z(n_z+1)/2) x 1}$ is a quadratic basis function. In literature [15][16][36], the quadratic basis vector $\phi(z_k)$ in LQR case is defined as a vector of quadratic terms of z_k

$$\phi(z_k) = z_k \otimes z_k = [{z_k}_1^2, {z_k}_1 {z_k}_2, \cdots, {z_k}_1 {z_k}_{n_z}, {z_k}_2^2, {z_k}_2 {z_k}_3, \cdots, {z_k}_2 {z_k}_{n_z}, \cdots, {z_k}_{n_z}^2]^T \quad (2.17)$$

where ${z_k}_{n_z}$ is the n_z[th] element of z_k.

The optimal policy is the policy that minimizes the optimal Q-function as in equation (2.9). In [16][17][36], the optimal policy minimizes (2.12). Without constraints this yields

$$\frac{\partial Q_h(x_k, u_k)}{\partial u_k} = 0 \quad (2.18)$$

The solution of this equation is presented combining $(2.11) - (2.13)$ as

$$u_k = h(x_k) = -{S_{uu}}^{-1} S_{ux}\, x_k. \quad (2.19)$$

Therefore, the optimal LQR gain is solved only using the measured states and control inputs without a system model.

2.1.2 Q-learning with output feedback measurements

Literature [15][33][34][36] derives a Q-learning method also for partially observable linear systems. First, the state x_k is denoted as

$$x_k = [M_u \quad M_y]\bar{x}_k \tag{2.20}$$

where the vector $\bar{x}_k$ is formed from previous controls and outputs as

$$\bar{x}_k = \begin{bmatrix}\bar{u}_k \\ \bar{y}_k\end{bmatrix}, \tag{2.21}$$

where $\bar{u}_k$ is a vector of old controls

$$\bar{u}_k = [\, u_{k-1} \quad u_{k-2} \quad \dots \quad u_{k-n}]^T \tag{2.22}$$

and $\bar{y}_k$ is a vector of old measurements

$$\bar{y}_k = [\, y_{k-1} \quad y_{k-2} \quad \dots \quad y_{k-n}]^T \tag{2.23}$$

and observability index n is chosen as $n \leq n_x$, where the upper bound n_x is the number of states. According to the articles, matrices M_u and M_y are given as

$$M_y = A^n (V_n^T V_n)^{-1} V_n^T \tag{2.24}$$

$$M_u = U_n - M_y T_n, \tag{2.25}$$

where the observability matrix V_n, controllability matrix U_n and Toeplizt matrix T_n are

$$V_n = [(CA^{n-1})^T \quad \cdots \quad (CA)^T \quad C^T]^T \tag{2.26}$$

$$U_n = [B \quad AB \quad \dots \quad A^{n-1}B] \tag{2.27}$$

$$T_n = \begin{bmatrix} 0 & CB & CAB & \cdots & CA^{n-2}B \\ 0 & 0 & CB & \cdots & CA^{n-3}B \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & CB \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.28}$$

Literature [15] defines the lower bound for observability index n as $n_k \leq n$. It is defined so that $\mathrm{rank}(V_n) < n_x$ when $n < n_k$, and $\mathrm{rank}(V_n) = n$ when $\mathrm{n} \geq n_k$. Therefore, n is selected as $n_k \leq n \leq n_x$.

The optimal policy for the output feedback Q-learning algorithms is derived in [15][33][34][36] by denoting the state x_k in equation (2.1) with the new notation (2.20). The new policy is given as

$$u_k = K^* x_k = K^* [M_u \quad M_y]\bar{x}_k \tag{2.29}$$

where K^* is the optimal full state control gain given in (2.4). The LQR state weighting parameter Q is calculated as

$$Q = C^T Q_y C\,, \tag{2.30}$$

where Q_y is the output weighting parameter. Instead of the full state x_k, Q-function uses only the knowledge of the state $\bar{x}_k$ and output y_k. Q-function is given in two forms as

$$Q_h(\bar{x}_k, u_k) = \bar{z}_k{}^T T \bar{z}_k = r(y_k, u_k) + \gamma \bar{z}_{k+1}{}^T T \bar{z}_{k+1}, \tag{2.31}$$

$$W^T\phi(\bar{z}_k) = r(y_k, u_k) + \gamma W^T\phi(\bar{z}_{k+1}) \tag{2.32}$$

where $\bar{z}_k \in \mathbb{R}^{n_{\bar{z}}}$, $n_{\bar{z}} = n(n_u + n_y) + n_u$, is defined with the new state $\bar{x}_k$ from (2.21) as

$$\bar{z}_k = \begin{bmatrix} \bar{x}_k \\ u_k \end{bmatrix} \tag{2.33}$$

and the new one-step cost $r(y_k, u_k)$ is

$$r(y_k, u_k) = {y_k}^T Q_y y_k + {u_k}^T R u_k. \tag{2.34}$$

and T is a symmetric matrix

$$T = \begin{bmatrix} T_{\bar{u}\bar{u}} & T_{\bar{u}\bar{y}} & T_{\bar{u}u} \\ T_{\bar{y}\bar{u}} & T_{\bar{y}\bar{y}} & T_{\bar{y}u} \\ T_{u\bar{u}} & T_{u\bar{y}} & T_{uu} \end{bmatrix} \tag{2.35}$$

where $T_{\bar{u}\bar{u}} \in \mathbb{R}^{nxn}$, $T_{\bar{u}\bar{y}} = {T_{\bar{y}\bar{u}}}^T \in \mathbb{R}^{nxn}$, $T_{\bar{u}u} = {T_{u\bar{u}}}^T \in \mathbb{R}^{nxn_u}$, $T_{\bar{y}\bar{y}} \in \mathbb{R}^{nxn}$, $T_{\bar{y}u} = {T_{u\bar{y}}}^T \in \mathbb{R}^{nxn_y}$ and $T_{uu} \in \mathbb{R}^{n_u x n_u}$ are the elements of matrix T. [15][33][34][36]

The new policy is derived in literature [15]-[36] as

$$u_k = h(\bar{x}_k) = -(T_{uu})^{-1}[T_{u\bar{u}} \quad T_{u\bar{y}}]\, \bar{x}_k. \tag{2.36}$$

This means that the optimal control is solved also without full state measurements.

2.1.3 Temporal difference based LQR Q-learning

Policy and value iteration are iterative temporal difference error based Q-learning algorithms for learning the optimal Q-function and control policy. Equation (2.11) can be expressed in a temporal difference error [16][17] form as

$$e = Q_h(x_k, u_k) - r(x_k, u_k) - Q_h(x_{k+1}, u_{k+1}). \tag{2.37}$$

Policy and value iteration iterate the Q-function $Q_h(x_k, u_k)$ and control policy u_k so that the temporal difference error e converges close to zero.

According to articles [16][17][36], policy and value iteration algorithms work forward-in-time and therefore they would be suitable for real-time control. The main difference between these two algorithms is that policy iteration is only used with a stabilizing initial control policy whereas value iteration can also be used without one. However, only policy iteration generates a stabilizing control each iteration whereas value iteration does not promise a stable control each iteration. For this reason, it is claimed in [12][25][36][43] that value iteration can be used safely only offline if the system itself is not stable.

Policy and value iteration algorithms repeat two update steps (see Figure 3 and Figure 4). The first step is often called policy evaluation or value update step in literature [15][17][36] and it updates the Q-function. The following step is called policy update and it updates the control policy based on the updated Q-function. In the linear case, policy

and value iteration Q-function is updated with different methods, such as least squares (LS), recursive least squares (RLS) or stochastic gradient descent (SGD) method. All of these can be on-policy (Figure 3) or off-policy methods (Figure 4).

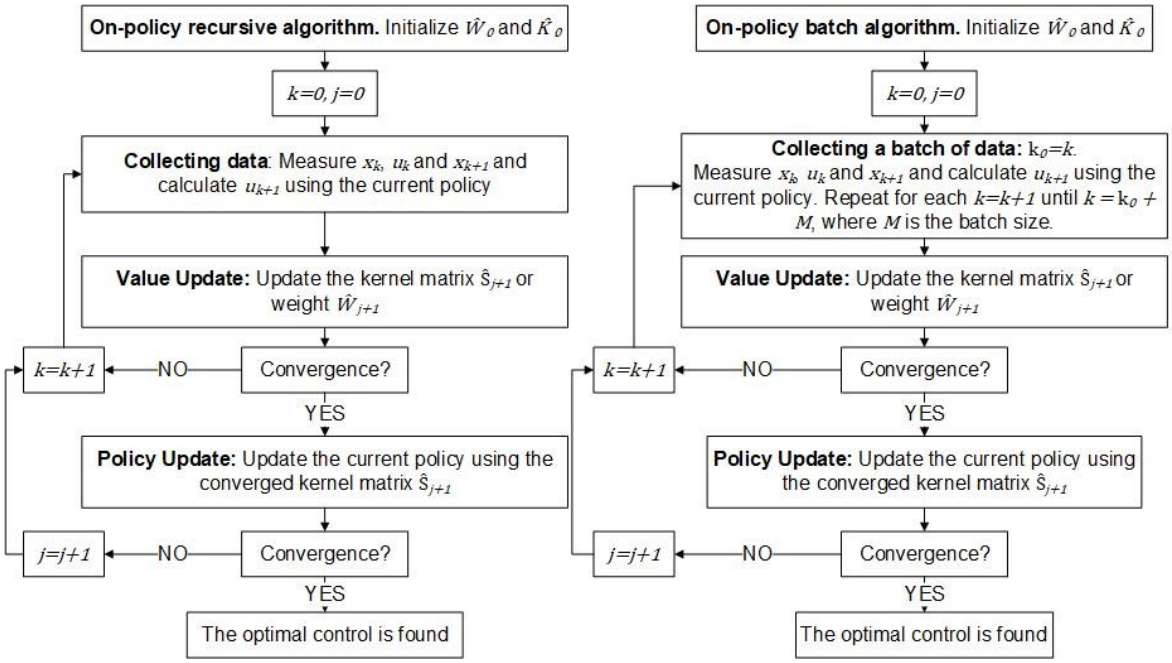

Figure 3. *On-policy value and policy iteration for LQR*

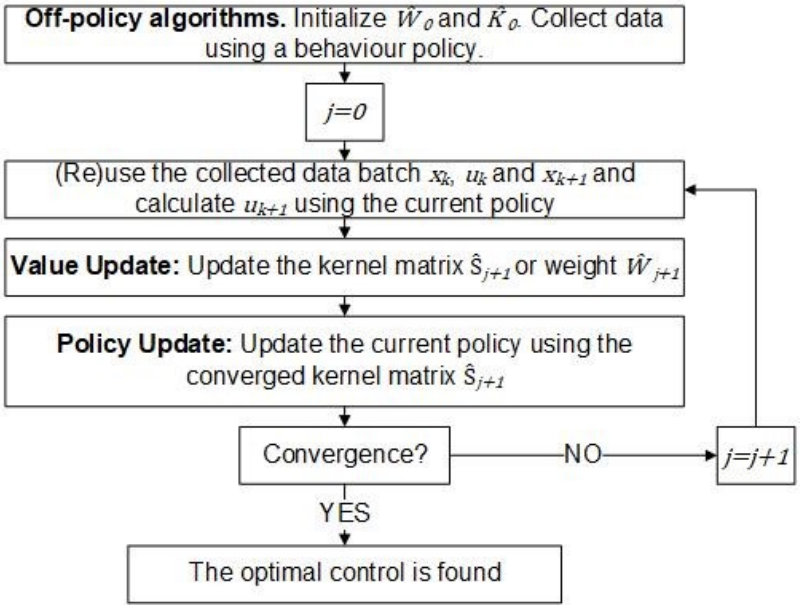

Figure 4. *Offline off-policy interleaved Q-learning for LQR*

Figure 4 shows an off-policy batch Q-learning algorithm. The method shown in Figure 4 is called policy and value iteration in articles [15][33]-[36], but interleaved Q-learning in [19][20] as the value update is done once unlike in (Figure 3). The more common naming is followed here and the off-policy algorithms are called policy and value iteration in this thesis.

2.1.4 Policy iteration (PI) equations for linear systems

Policy iteration follows the Figure 3 or Figure 4 procedure. It is known in literature (e.g. [13][15]-[17]) that this algorithm needs a stabilizing initial gain. This can be found with some initial knowledge of the system such as system operators experiences as was mentioned in [13].

The initial control gain K_0 must be stabilizing, but the initial kernel matrix $\hat{S}_0$ can be chosen randomly. The algorithm is initialized at $j = 0$. Generally policy iteration value update is given in [15]-[17][42] as

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_{j+1}\left(x_{k+1}, h_j(x_{k+1})\right) \tag{2.38}$$

which makes the temporal difference error

$$e_j = Q_{j+1}(x_k, u_k) - r(x_k, u_k) - Q_{j+1}(x_{k+1}, u_{k+1}) \tag{2.39}$$

And the policy update can be as given in [16][17][25] as

$$h_{j+1}(x_k) = \arg\min_{u_k}\left(Q_{j+1}(x_k, u_k)\right) \tag{2.40}$$

Policy iteration algorithm for LQR can be derived combining (2.39) with (2.12) or (2.15). The value update is given in [16][17][36] either using the kernel matrix $\hat{S}_{j+1}$ given as

$$z_k{}^T \hat{S}_{j+1} z_k - \gamma z_{k+1}{}^T \hat{S}_{j+1} z_{k+1} = r(x_k, u_k) \tag{2.41}$$

or using the weight matrix $\widehat{W}_{j+1}$ given as

$$\widehat{W}_{j+1}^T \varphi_k = \mu_k \,, \tag{2.42}$$

where the right-hand side μ_k, the data vector, is now

$$\mu_k = r(x_k, u_k) \tag{2.43}$$

and the left-hand term φ_k of (2.42), the regression vector, is

$$\varphi_k = \phi(z_k) - \gamma\phi(z_{k+1}) \tag{2.44}$$

where $\phi_k(z_k)$ is the basis vector given in (2.17) and z_k vector given in (2.13). Depending on which notation, (2.41) or (2.42), is used, either the kernel matrix $\hat{S}_{j+1}$ or the weight matrix $\widehat{W}_{j+1}$ is approximated during the value update from one of these equations. Q-function value is updated by either least squares, recursive least squares or stochastic gradient descent method until the matrix converges.

As seen in Figure 3, the value is updated until convergence with growing time index k and the kernel matrix $\hat{S}_{j+1}$ or the weight matrix $\widehat{W}_{j+1}$ is then used for policy update given in (2.19). If the matrix $\widehat{W}_{j+1}$ is used, it needs to be unpacked into matrix $\hat{S}_{j+1}$ with the knowledge of (2.14) and (2.16) before the policy update. The updated policy is used for

new value update and these two steps are repeated until the weight converges, so that $\|\widehat{W}_{j+1} - \widehat{W}_j\| \leq \varepsilon_j$, where ε_j is a small constant.

If only output measurements are known, the equations (2.41) or (2.42) are replaced by the equivalent equations derived by using (2.31) or (2.32) as

$$\begin{cases} \bar{z}_k{}^T \hat{T}_{j+1} \bar{z}_k - \gamma \bar{z}_{k+1}{}^T \hat{T}_{j+1} \bar{z}_{k+1} = r(x_k, u_k) \\ \widehat{W}_{j+1}^T \varphi_k = \mu_k \\ \mu_k = r(y_k, u_k) \\ \varphi_k = \phi(\bar{z}_k) - \gamma \phi(\bar{z}_{k+1}) \end{cases} \tag{2.45}$$

and policy update (2.19) is replaced by equation (2.36) [15][36].

According to [15][16][23][25], to get linearly independent data and to ensure the persistence of excitation (PE) condition and the convergence of the kernel matrix $\hat{S}_{j+1}$, an exploration noise ϵ_k is added to the control input. Many articles [33]-[36] select the exploration noise as random Gaussian noise or sum of sine waves of different frequencies. The discounting factor γ can be adjusted to remove bias from the solution by setting $0 < \gamma < 1$. However, the article in [36] proves that choosing $\gamma < 1$, as in [15], makes the quadratic cost function (2.2) finite, but the closed-loop system stability is not guaranteed.

2.1.5 Replacing policy iteration with value iteration (VI)

Value iteration algorithm follows the procedure in Figure 3 or Figure 4. The algorithm structure is the same as described in the Chapter 2.1.4 for policy iteration, but the value update step is changed, and the algorithm needs no stabilizing policy. The value update in value iteration algorithms is generally given in [16][17][43] as

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_j \left(x_{k+1}, h_j(x_{k+1}) \right) \tag{2.46}$$

making the temporal difference error

$$e_j = Q_{j+1}(x_k, u_k) - r(x_k, u_k) - Q_j(x_{k+1}, u_{k+1}). \tag{2.47}$$

Therefore equation (2.41) is replaced by

$$z_k{}^T \hat{S}_{j+1} z_k = r(x_k, u_k) + \gamma z_{k+1}{}^T \hat{S}_j z_{k+1} \tag{2.48}$$

and the vectors (2.43) and (2.44) in equation (2.42) are replaced by vectors

$$\varphi_k = \phi(z_k) \tag{2.49}$$

$$\mu_k = r(x_k, u_k) + \gamma \widehat{W}_j{}^T \phi(z_{k+1}), \tag{2.50}$$

where $\hat{S}_j$ and $\widehat{W}_j$ are the kernel and weight matrices from the previous iteration j.

If full states are unknown, equation (2.45) is replaced by the following equations [15][36].

$$\begin{cases} \bar{z}_k{}^T \hat{T}_{j+1} \bar{z}_k = r(y_k, u_k) + \gamma \bar{z}_{k+1}{}^T \hat{T}_{j+1} \bar{z}_{k+1} \\ \widehat{W}_{j+1}^T \varphi_k = \mu_k \\ \mu_k = r(y_k, u_k) + \gamma \widehat{W}_j^T \phi(\bar{z}_{k+1}) \\ \varphi_k = \phi(\bar{z}_k) \end{cases} \tag{2.51}$$

An exploration noise ϵ_k is added to the control input to ensure exploration of the signal.

2.1.6 Batch least squares weight $\widehat{W}_{j+1}$ value update for PI and VI

Batch least squares (LS) method is one of the four discussed methods (see Figure 1) that could be used to calculate the policy and value iteration value update as seen in Figure 3 and Figure 4. This algorithm updates the weight matrix $\widehat{W}_{j+1}$ based on equation (2.42). The algorithm fits $\widehat{W}_{j+1}$ so that temporal difference error is becomes small [15][36]. When value iteration is used, the matrix $\widehat{W}_j$ is known from previous iterations.

The weight update (2.42) is modified in [15][36] for a batch algorithm as $\widehat{W}_{j+1}^T \Phi = Y$, where regression matrix $\Phi \in \mathbb{R}^{(n_z(n_z+1)/2)xM}$ and data matrix $Y \in \mathbb{R}^{(n_z(n_z+1)/2)x1}$ are formed with the regression vectors φ_k and data vectors μ_k as

$$\Phi = [\varphi_k, \varphi_{k+1}, \dots, \varphi_{k+M}] \tag{2.52}$$

$$Y = [\, \mu_k, \mu_{k+1}, \dots, \mu_{k+M}]^T \tag{2.53}$$

where M is the batch size. According to literature [15][36] the batch size M is $M \geq n_z(n_z+1)/2$. To generate the matrices Y and Φ, data and regression vectors of choice are chosen from equations $(2.43) - (2.44)$, $(2.49) - (2.50)$, (2.45) or (2.51) depending on which algorithm is used and if full states are known. These vectors use the measurements x_k, u_k and x_{k+1} and u_{k+1} calculated with (2.19) when full states are known. If only output measurements are known, for each data point the vectors $\bar{x}_k$ and $\bar{x}_{k+1}$ are calculated with $(2.21 - 2.23)$ and $u_{k+1}$is calculated using (2.36).

The least squares update in [6][15][17][36] solves one-step weight update as

$$\widehat{W}_{j+1} = (\Phi\Phi^T)^{-1}\Phi\mathrm{Y}. \tag{2.54}$$

The inverse in (2.54) exists only if exploration noise is added to the control so that $\mathrm{rank}(\Phi) = n_z(n_z+1)/2$. Lastly, the elements of $\widehat{W}_{j+1}$ are unpacked into an updated kernel matrix $\hat{S}_{j+1}$ or $\hat{T}_{j+1}$.

2.1.7 Batch least squares kernel matrix $\widehat{S}_{j+1}$ value update for PI and VI

This is the second discussed method for the policy and value iteration value update (see Figure 1 and Figure 4). The kernel matrix $\hat{S}_{j+1}$ is updated without a basis function using equation (2.41) or (2.48). When value iteration is used, the previously updated matrix $\hat{S}_j$ is known from previous iterations.

This is a batch value update. The temporal difference error e_j at step j given in (2.39) and (2.47) is changed for policy iteration as

$$e_{LS,j} = Z_k^T \hat{S}_{j+1} Z_k - \Psi_k - \gamma Z_{k+1}^T \hat{S}_{j+1} Z_{k+1} \tag{2.55}$$

$$\Psi_k = X_k^T Q X_k + U_k^T R U_k. \tag{2.56}$$

and for value iteration as

$$e_{LS,j} = Z_k^T \hat{S}_{j+1} Z_k - \Psi_k \,. \tag{2.57}$$

$$\Psi_k = X_k^T Q X_k + U_k^T R U_k + \gamma Z_{k+1}^T \hat{S}_j Z_{k+1}. \tag{2.58}$$

The batch size M is $M \geq n_z(n_z+1)/2$ [15][36]. For each time k in the batch M, the states x_k and x_{k+1} and control u_k are measured, u_{k+1} is calculated using equation (2.19) and matrices $\mathrm{z_k}$ and $\mathrm{z_{k+1}}$ are calculated with (2.13). This data is collected into matrices $X_k \in \mathbb{R}^{n_x xM}$, $U_k \in \mathbb{R}^{n_u xM}$, $\mathrm{Z_k} \in \mathbb{R}^{n_z xM}$ and $\mathrm{Z_{k+1}} \in \mathbb{R}^{n_z xM}$ as follows

$$X_k = [x_k, x_{k+1}, \dots, x_{k+M}] \tag{2.59}$$

$$U_k = [u_k, u_{k+1}, \dots, u_{k+M}] \tag{2.60}$$

$$\mathrm{Z_k} = [z_k, z_{k+1}, \dots, z_{k+M}], \mathrm{Z_{k+1}} = [z_{k+1}, z_{k+2}, \dots, z_{k+M+1}] \tag{2.61}$$

If only output measurements are known, the vectors $\bar{x}_k$ and $\bar{x}_{k+1}$ are formed using equation (2.21) and u_{k+1} is calculated using (2.36) and $\bar{\mathrm{z}}_\mathrm{k}$ and $\bar{\mathrm{z}}_\mathrm{k+1}$ are formed as in (2.33). Vectors $\bar{x}_k$ and $\bar{\mathrm{z}}_\mathrm{k}$ replace the equivalent elements in the batch matrices $(2.59) - (2.61)$.

The kernel matrix $\hat{S}_{j+1}$ can be fitted with least squares. Equation (2.47) for value iteration is now a batch equation

$$Z_k^T \hat{S}_{j+1} Z_k = \Psi_k \,, \tag{2.62}$$

where the right-hand side was given in (2.58). Equation (2.62) derived as

$$\Phi \hat{S}_{j+1} = Y \tag{2.63}$$

$$\begin{cases} Y = \Psi_k Z_k^T \left(Z_k Z_k^T\right)^{-1} \\ \Phi = Z_k^T \end{cases} \tag{2.64}$$

Matrix Y exists if matrix $Z_k Z_k^T$ is invertible. Matrix $Z_k Z_k^T$ is invertible if $\mathrm{rank}(Z_k) = n_z$ [15][17][36][42]. This condition is satisfied with exploration noise added to the control. Since the kernel matrix $\hat{S}_{j+1}$is symmetric, equation (2.41) for batch policy iteration is given as

$$Z_k^T \hat{S}_{j+1} Z_k - \sqrt{\gamma} Z_{k+1}^T \hat{S}_{j+1} \sqrt{\gamma} Z_{k+1} = \left(Z_k^T - \sqrt{\gamma} Z_{k+1}^T\right) \hat{S}_{j+1} \left(Z_k + \sqrt{\gamma} Z_{k+1}\right) = \Psi_k \tag{2.66}$$

The least squares matrices are Y and Φ in (2.65) are

$$\begin{cases} Y = \Psi_k (Z_k + \sqrt{\gamma} Z_{k+1})^T \left((Z_k + \sqrt{\gamma} Z_{k+1})(Z_k + \sqrt{\gamma} Z_{k+1})^T\right)^{-1} \\ \Phi = \left(Z_k^T - \sqrt{\gamma} Z_{k+1}^T\right) \end{cases} \tag{2.67}$$

And matrix $(Z_k + \sqrt{\gamma} Z_{k+1})(Z_k + \sqrt{\gamma} Z_{k+1})^T$ must be invertible to solve Y. Matrix $(Z_k + \sqrt{\gamma} Z_{k+1})(Z_k + \sqrt{\gamma} Z_{k+1})^T$ is invertible if $\mathrm{rank}(Z_k + \sqrt{\gamma} Z_{k+1}) = n_z$.

Similarly as in (2.54) and in [6][15][17][36] the kernel matrix $\hat{S}_{j+1}$ can be updated with least squares using an update formula

$$\hat{S}_{j+1} = (\Phi^T \Phi)^{-1} \Phi^T \mathrm{Y}, \tag{2.65}$$

where $\Phi^T \Phi$ is invertible if $\mathrm{rank}(\Phi) = n_z$. With value iteration, calculation rules lead to $\mathrm{rank}(\Phi) = \mathrm{rank}\left(Z_k^T\right) = \mathrm{rank}(Z_k)$. Therefore, both rank conditions are satisfied simultaneously. For policy iteration the rank conditions are not equal and both must be satisfied separately.

This value update can also be calculated using nonlinear least squares. The objective in this case is to find a kernel matrix $\hat{S}_{j+1}$ that minimizes the mean square error

$$\min_{\hat{S}_{j+1}} \left(\frac{1}{2} {e_{LS,j}}^T e_{LS,j}\right) \tag{2.68}$$

The unknown kernel matrix $\hat{S}_{j+1}$ is solved using equation (2.55) or (2.57) by starting from an initial guess $\hat{S}_0$ and iterating until the minimum of the mean square error is found. [9]

2.1.8 Recursive least squares value update for PI and VI

Recursive least squares (RLS) value update is a third discussed method for value update (see Figure 1 and Figure 3). Each time step a new weight $\widehat{W}_{j+1,i}$ is iterated until it converges and the value $\widehat{W}_{j+1,\infty}$ is updated.

Firstly, the initial value for covariance matrix P_0 is chosen as

$$P_0 = \delta I \tag{2.69}$$

where δ is a large scalar [6]. The index i is initialized as $i = 0$ and $\widehat{W}_{j+1,0} = \widehat{W}_j$.

If full states are known, x_k, x_{k+1} and u_k are measured at time k and u_{k+1} is calculated using equation (2.19) as in [16][17]. Otherwise, vectors $\bar{x}_k$ and $\bar{x}_{k+1}$ are formed using (2.21) and $u_{k+1}$is calculated using (2.36) as in [15][33]-[36]. Depending on which algorithm is chosen, the regression φ_k and data vector μ_k are chosen from $(2.43) - (2.44)$, $(2.49) - (2.50)$, (2.45) or (2.51).

As given in literature $[6]$, during the one-step update of the weight $\widehat{W}_{j+1,i}$, the update matrix L_{i+1} is calculated as

$$L_{i+1} = \lambda^{-1} P_i \varphi_k \left(a^{-1} + \lambda^{-1} \varphi_k^T P_i \varphi_k\right)^{-1}, \tag{2.70}$$

where P_i is the covariance matrix at iteration i and λ is a recursive least squares discounting factor. For regular least squares $\lambda = 1$ and $a = 1$ and for exponentially weighted recursive least squares $0 < \lambda < 1$ and $a = 1 - \lambda$. The weight matrix $\widehat{W}_{j+1,i}$ and the covariance matrix P_{i+1} are updated with

$$\widehat{W}_{j+1,i+1} = \widehat{W}_{j+1,i} + L_{i+1}\left(\mu_k - \varphi_k^T \widehat{W}_{j+1,i}\right) \tag{2.71}$$

$$P_{i+1} = \lambda^{-1}\left(I - L_{i+1}\varphi_k^T\right)P_i \tag{2.72}$$

At the next time $k+1$ and next iteration $i+1$, new measurements are taken with the current policy. The one-step updates of equations $(2.70) - (2.72)$ are repeated with new data until converge so that $\left\|\widehat{W}_{j+1,i+1} - \widehat{W}_{j+1,i}\right\| \leq \varepsilon_i$, where ε_i is a small constant. The converged weight $\widehat{W}_{j+1,\infty}$ is denoted shortly as $\widehat{W}_{j+1} = \widehat{W}_{j+1,\infty}$, which is the updated value.

2.1.9 Stochastic gradient descent value update for PI and VI

Fourth discussed method for value update is stochastic gradient descent (SGD) (see Figure 3). It fits linear models with a small a batch of data, if the data samples in these mini-batches are independent from each other [7]. The weight $\widehat{W}_{j+1,i}$ is kept training with growing time k until convergence. The value update step is initialized with $i = 0$ and $\widehat{W}_0 = \widehat{W}_j$.

If full states are known, x_k and x_{k+1} and u_k are measured and u_{k+1} is calculated using equation (2.19) [16][17]. Otherwise, vectors $\bar{x}_k$ and $\bar{x}_{k+1}$ are formed using (2.21) and u_{k+1} is calculated using (2.36) [15][33]-[36]. The regression and data vectors φ_k and μ_k are chosen from equations $(2.43) - (2.44)$, $(2.49) - (2.50)$, (2.45) or (2.51) using the measured data. The temporal difference error from (2.39) and (2.47) is given as

$$e_{sgd,i} = \widehat{W}_{j+1,i}^T \varphi_k - \mu_k \tag{2.74}$$

The mean square error to be minimized is given as

$$E_{sgd,i} = \frac{1}{2} {e_{sgd,i}}^T e_{sgd,i} \tag{2.75}$$

Its gradient $\frac{\partial E_{sgd,i}}{\partial \widehat{W}_{j+1,i}}$ is calculated using the chain rule. For policy iteration it is given as

$$\frac{\partial E_{sgd}}{\partial \widehat{W}_{j+1,i}} = \frac{\partial E_{sgd}}{\partial e_{sgd,i}} \frac{\partial e_{sgd,i}}{\partial \widehat{W}_{j+1,i}} = \big(\phi(z_k) - \gamma\phi(z_{k+1})\big) {e_{sgd,i}}^T \tag{2.76}$$

and for value iteration it is similarly

$$\frac{\partial E_{sgd,i}}{\partial \widehat{W}_{j+1,i}} = \phi(z_k) {e_{sgd,i}}^T. \tag{2.77}$$

The one-step weight update is derived in literature [23] as

$$\widehat{W}_{j+1,i+1} = \widehat{W}_{j+1,i} - \alpha_{sgd} \frac{\partial E_{sgd}}{\partial \widehat{W}_{j+1,i}} \tag{2.78}$$

where the learning rate α_{sgd} is a constant.

New measurements are taken with the current policy at the next time k and next iteration i and the weight (2.78) is updated with the new data. These updates are repeated until $\left\|\widehat{W}_{j+1,i+1} - \widehat{W}_{j+1,i}\right\| \leq \varepsilon_i$, where ε_i is a small constant. After convergence, the new value $\widehat{W}_{j+1} = \widehat{W}_{j+1,\infty}$ is found.

2.2 Reinforcement Q-learning in nonlinear model-free control

Systems are generally nonlinear and the mathematical models can be complex, therefore manual identification can be difficult [6][37]. Model-free optimal control for nonlinear systems is solved with adaptive dynamic programming. So far, most nonlinear ADP algorithms are VFA algorithms. VFA with nonaffine discrete-time systems is discussed in [13][21][23][25][40][41][43] and with affine discrete-time systems in [2]-[4][10][11][24][38]. Nonetheless, there are some Q-learning applications with nonlinear systems. Q-learning with nonaffine discrete-time systems is derived in [22][26][31][27][42] and with affine discrete-time systems [19][44].

According to [16], control constraints make it difficult to solve the control problem with policy and value iteration algorithms for nonlinear systems. Therefore, policy iteration in [42] is not used. Nonaffine Q-learning articles [31][27][26] derive new nonlinear Q-learning methods called NFQCDA (Neural Fitted Q-Learning with Continuous Discrete Actions), PGADP (Policy Gradient Adaptive Dynamic Programming) and CoQL (Critic-only Q-learning). These methods have not been studied in much detail yet, whereas interleaved Q-learning for affine systems in [19] has been already studied for VFA applications in [13] and [23]. Interleaved Q-learning in [19] is studied in more detail here over the other methods as it is gives proof that the estimations are not biased due to the

exploration noise. The method is also called simultaneous, time-based or synchronous ADP in [19][23][32].

Nonlinear affine discrete-time system model can be expressed as

$$\begin{cases} x_{k+1} = f(x_k) + g(x_k)u_k \\ y_k = c(x_k) \end{cases}, \tag{2.79}$$

where $f(x_k)$ includes the inner dynamics of the system, $g(x_k)$ includes the input dynamics and $c(x_k)$ includes the output dynamics [6][19].

To find the optimal control for the nonlinear affine discrete-time system in (2.79), the performance index must be minimized as it was minimized in Chapter 2.1 for linear systems. In [19], the performance index is given as

$$V_h(x_k) = \sum_{k=0}^{\infty} r(x_k, u_k) \tag{2.80}$$

and $r(x_k, u_k)$ is given in (2.3) with $i_k = k$. The equation set (2.7) can be applied now with $\gamma = 1$. The optimal value function in (2.7) is also the optimal value of the Hamilton-Jacobi-Bellman (HJB) equation. HJB equation cannot be solved analytically for affine nonlinear system and therefore it is approximated. The optimal policy $h^*(x_k)$ in (2.7) is calculated in [19] as

$$u_k^* = -\frac{1}{2} R^{-1} g(x_k)^T \frac{\partial V^*(x_{k+1})}{\partial x_{k+1}} \tag{2.81}$$

Using $(2.8 - 2.9)$ the equations (2.7) and (2.81) for nonlinear Q-learning are derived as

$$\begin{cases} Q^*(x_k, u_k) = r(x_k, u_k) + \gamma Q^*(x_{k+1}, u_{k+1}^*) \\ u_k^* = -\frac{\gamma}{2} R^{-1} g(x_k)^T \frac{\partial Q^*(x_{k+1}, u_{k+1}^*)}{\partial x_{k+1}} \end{cases} \tag{2.82}$$

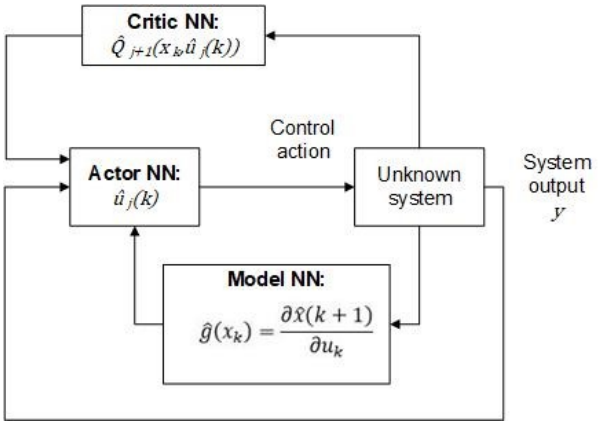

Figure 5. *Actor-critic architecture with added model network*

The chosen approach follows the actor-critic architecture of Figure 5 to approximate the optimal HJB solution of (2.82). The actor is a neural network that learns the optimal policy

and the critic is a network that learns the optimal Q-function. Later it is shown that the actor network needs knowledge of the matrix $g(x_k)$. An additional network is needed to identify this matrix to make the algorithm model-free. This network is called model neural network or identification network and it is commonly used with VFA in literature [3][13][19][23][25][37][40].

2.2.1 Off-policy interleaved Q-learning with full state measurements

Interleaved Q-learning is a neural batch fitted Q-learning method for learning the optimal control for affine nonlinear systems as derived in [13][19][23]. The HJB equation in (2.82) is solved approximately by three networks that are simultaneously updated. The goal is to minimize the learning error of each network by minimizing the mean square error $E_{nw}(k)$ of each network nw. The mean square error is defined by

$$E_{nw,j}(k) = \frac{1}{2} e_{nw,j}^T(k) e_{nw,j}(k) \tag{2.83}$$

where $e_{nw}(k)$ is the used network estimation error at time k. This is minimized by using gradient descent based methods to update the network weight $\widehat{W}_{nw,j+1}$.

A weight update at iteration j for the network weight $\widehat{W}_{nw,j+1}$ is given as

$$\widehat{W}_{nw,j+1} = \widehat{W}_{nw,j} - \alpha_{nw} \frac{\partial E_{nw,j}(k)}{\partial \widehat{W}_{nw,j}(k)}, \tag{2.84}$$

where $\widehat{W}_{nw,j+1}$ and $\widehat{W}_{nw,j}$ are the weights at time k, nw is the used network, α_{nw} is the learning rate and $\frac{\partial E_{nw,j}(k)}{\partial \widehat{W}_{nw,j}(k)}$ is the gradient of the mean square error of the network.

The output of each network is estimated using the learned weights $\widehat{W}_{nw,j+1}$ and an activation function. Many articles [13][19][23] choose the activation function $\sigma(z)$ for each network as a $\tanh$-function so that one neuron i_{nw} is

$$[\sigma(z)]_{i_{nw}} = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.85}$$

and there are n_{nw} is the number of neurons. Its derivative is

$$[\dot{\sigma}(z)]_{i_{nw}} = 1 - \tanh^2(z) \tag{2.86}$$

Off-policy neural batch fitted Q-learning methods [8][19][31] learn the optimal policy by using a set of training data that is collected in using a stabilizing behaviour policy with added exploration noise. Collecting data is the first step of the algorithm as shown in Figure 6. Training data is collected from several experiments as in [32][42] within the

control region. The experiment is repeated for n_{sets} times starting from the same initial state until N time steps. These learning episodes are divided into N sample sets so that time steps k from each learning episode form their own sample batches of n_{sets} items. In [32], a larger number of learning episodes leads to better success rates, when the success was measured as the number of successful learning trials out of 50 samples.

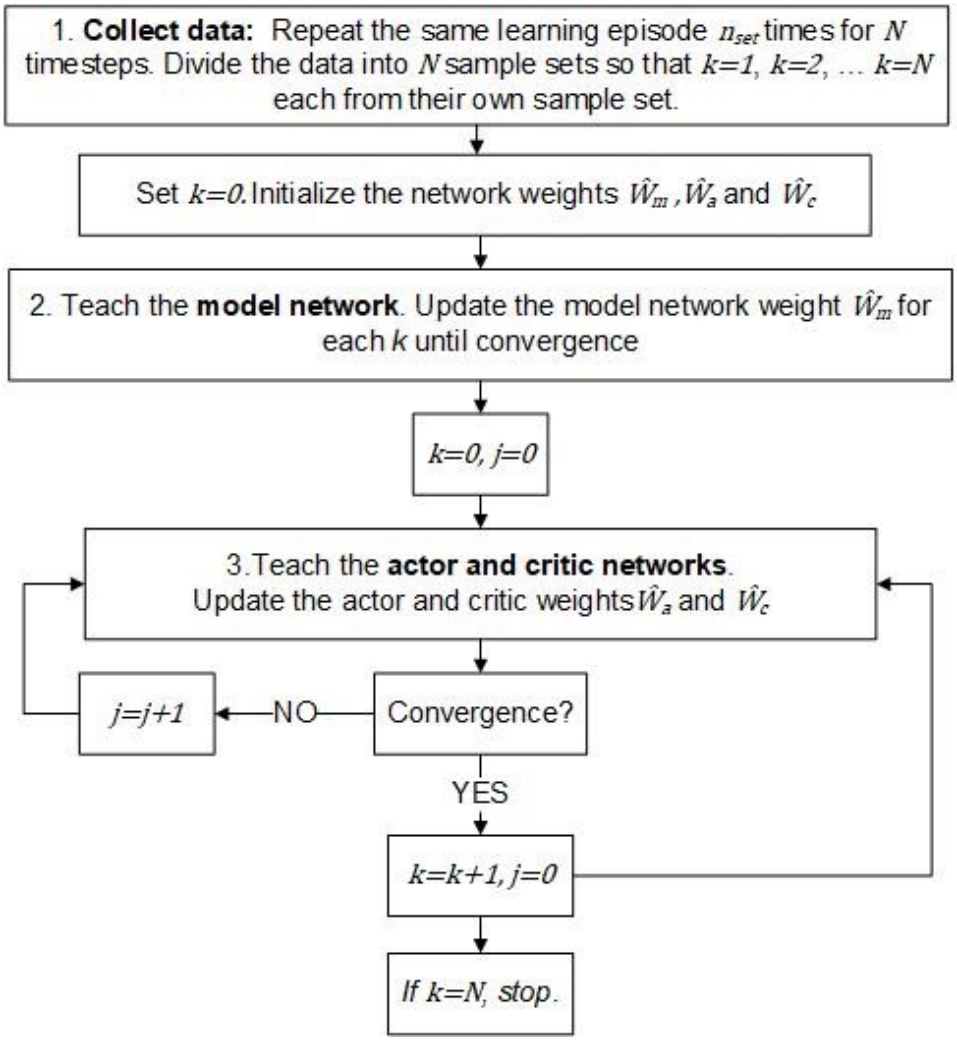

Figure 6. *Interleaved Q-learning procedure*

Interleaved Q-learning in [19] is a value iteration based method and therefore the initial weights do not have to be stabilizing. The weights $\widehat{W}_{a,0}$, $\widehat{W}_{c,0}$ and $\widehat{W}_{m,0}$ of the three networks are initialized randomly before the algorithm. As Figure 6 shows, the whole data is used to learn the model neural network weights. The time index is initialized at $k = 0$ and the iteration j is not used yet. The model network output $\hat{x}_{k+1}$ is given in [3][13][19][23][25][37][40] as

$$\hat{x}_{k+1} = \widehat{W}_m^T \sigma\left(v_m^T \begin{bmatrix} x_k \\ u_k \end{bmatrix}\right) \tag{2.87}$$

where $\widehat{W}_m(k)$ is the model network weight matrix and v_m is the constant model network

hidden layer weight matrix. The model network can estimate the input dynamics $\hat{g}(x_k)$. Input dynamics are calculated using equation (2.79) and taking the gradient of $\hat{x}(k+1)$ in terms of u_k so that

$$\frac{\partial \hat{x}_{k+1}}{\partial u_k} = \hat{g}(x_k). \tag{2.88}$$

The estimation error of the model network is

$$e_m(k) = \hat{x}_{k+1} - x_{k+1} \tag{2.89}$$

The mean square error $E_m(k)$ of the model network is calculated using (2.83) when $nw = m$ and its gradient $\frac{\partial E_m(k)}{\partial \widehat{W}_m(k)}$ is derived as.

$$\frac{\partial E_m(k)}{\partial \widehat{W}_m(k)} = \sigma\left(v_x^T \begin{bmatrix} x_k \\ u_k \end{bmatrix}\right) e_m^T(k) \tag{2.90}$$

The model network is updated with the gradient descent update when $nw = m$ is modified from (2.81) as

$$\widehat{W}_m(k+1) = \widehat{W}_m(k) - \alpha_{nw} \frac{\partial E_m(k)}{\partial \widehat{W}_m(k)} \tag{2.91}$$

Equations $(2.87 - 2.91)$ are repeated for each k until the estimation error $\|e_x(k)\| \leq \varepsilon_m$, where ε_m is a small constant. The learned $\hat{x}_{k+1}$ is used inside the interleaved Q-learning algorithm.

After the model network convergence, the critic and the actor networks are updated offline simultaneously as shown in Figure 6. The algorithm is initialized with $k = 0$. For each time k, the Q-function is initialized at $j = 0$ as $Q_0(\cdot) = 0$ and the initial target policy $u_0(x_k)$ is calculated using

$$u_0(x_k) = \arg\min_{u_k} \left(x_k^T Q x_k + u_k^T R u_k + Q_0(\cdot)\right). \tag{2.92}$$

The Q-function is iterated with growing index j until convergence for each time k [13][19].

The temporal difference error, or the critic network estimation error, is given in [19] as

$$\begin{aligned} e_{c,j+1}(k) &= \hat{Q}_{j+1}\left(x_k, \hat{u}_j(x_k)\right) - Q_{j+1}\left(x_k, \hat{u}_j(x_k)\right) \\ &= \hat{Q}_{j+1}\left(x_k, \hat{u}_j(x_k)\right) - r\left(x_k, \hat{u}_j(x_k)\right) - \hat{Q}_j\left(x_{(k+1),j}, \hat{u}_j(x_k)\right), \end{aligned} \tag{2.93}$$

where the critic network output is

$$\hat{Q}_{j+1}\left(x_k, \hat{u}_j(x_k)\right) = \widehat{W}_{c,j+1}^T(k)\sigma\left(v_c^T \begin{bmatrix} x_k \\ \hat{u}_j(x_k) \end{bmatrix}\right) \tag{2.94}$$

$$\hat{Q}_j\left(x_{(k+1),j}, \hat{u}_j\left(x_{(k+1),j}\right)\right) = \widehat{W}_{c,j}^T(k)\sigma\left(v_c^T \begin{bmatrix} x_{(k+1),j} \\ \hat{u}_j\left(x_{(k+1),j}\right) \end{bmatrix}\right) \tag{2.95}$$

$$x_{(k+1),j} = x_{k+1} - \hat{g}(x_k)\left(u_k - \hat{u}_j(x_k)\right). \tag{2.96}$$

and $\widehat{W}_c(k)$ is the critic network weight matrix, v_c is the constant critic network hidden layer weight matrix and $\hat{u}_j(x_k)$ and $\hat{u}_j\left(x_{(k+1),j}\right)$ are the actor network outputs in $(2.98 - 2.99)$. The mean square error is calculated using (2.83) when $nw = c$ and its gradient for the critic is calculated as in [13][19][23][42] as in (2.77) as

$$\frac{\partial E_{c,j}(k)}{\partial \widehat{W}_{c,j}(k)} = \sigma\left(v_c^T \begin{bmatrix} x_k \\ \hat{u}_j(x_k) \end{bmatrix}\right) e_{c,j}^T(k)\,. \tag{2.97}$$

and the network weights are updated with (2.84) when $nw = c$.

The actor network estimates the control policy

$$\hat{u}_j(x_k) = \widehat{W}_{a,j}^T \sigma(v_a^T x_k) \tag{2.98}$$

$$\hat{u}_j\left(x_{(k+1),j}\right) = \widehat{W}_{a,j}^T \sigma\left(v_a^T x_{(k+1),j}\right) \tag{2.99}$$

where $\widehat{W}_a(k)$ and v_a are the actor network weight and constant hidden layer weight matrices. The estimation error of the actor network is derived in [19][23] as

$$e_{a,j}(k) = \hat{u}_j(x_k) - u_j(x_k), \tag{2.100}$$

where $u_j(k+1)$ is called a target control policy. For affine nonlinear systems [16][17] the target policy is

$$u_j(x_k) = -\frac{\gamma}{2} R^{-1} \hat{g}(x_k)^T \frac{\partial \hat{Q}_j\left(x_{(k+1),j}, \hat{u}_j\left(x_{(k+1),j}\right)\right)}{\partial\left(x_{(k+1),j}\right)} \tag{2.101}$$

The mean square error is calculated with (2.83) when $nw = a$ and its gradient for the actor network is

$$\frac{\partial E_{a,j}(k)}{\partial \widehat{W}_{a,j}(k)} = \sigma(v_a^T x_k) e_{a,j}^T(k) \tag{2.102}$$

The network weights are updated with (2.81), when $nw = a$.

Equations $(2.92) - (2.102)$ are repeated as the Figure 6 suggests with growing index j until $\left\| \hat{Q}_j\left(x_k, \hat{u}_{j-1}(k)\right) - \hat{Q}_{j+1}\left(x_k, \hat{u}_j(k)\right) \right\| \leq \varepsilon_j$, where ε_j is constant. After convergence, the optimal policy $\hat{u}_j(k)$ for time k is saved. These interleaved iterations are repeated for each k starting from $j = 0$ until all networks have converged. [19]

3. IMPLEMENTING MODEL-FREE CONTROL

Model-free control is implemented in linear and nonlinear real-time systems using different on-policy and off-policy methods that were shown in Figure 1. Implementation in this thesis is stepwise (see Figure 7). The first step is to develop a reference point. The selected algorithms are implemented with a theoretical system model. Both the full state and output feedback versions of the algorithms are implemented. The model is used for data generation only. Next, the theoretical model is modified to resemble the real-time system better. Lastly, the theoretical model is replaced with the real-time system.

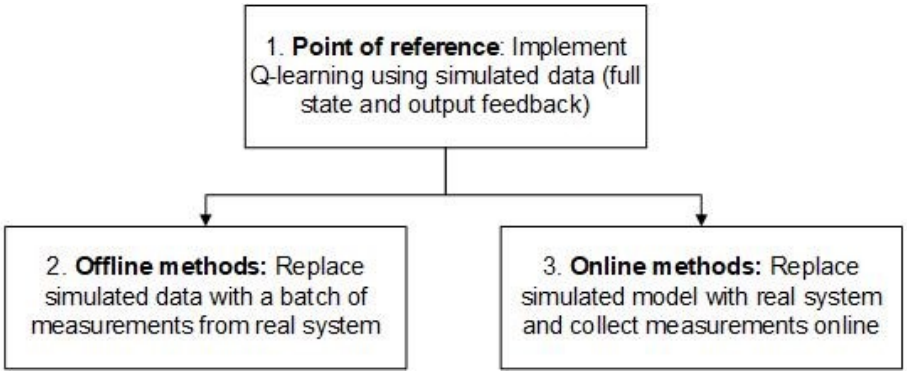

Figure 7. *Model-free Q-learning implementation steps in this thesis*

The real-time system in this study is a Quanser QUBE™- Servo 2 experiment. It has two attachments, inertia disk and inverted pendulum.

3.1 Laboratory setting and environments

The Quanser QUBE™-Servo 2 rotary servo experiment is compatible with MATLAB and LabVIEW [1][30]. MATLAB was chosen as the platform in this study and Visio 2013 Professional as the C++ compiler for MATLAB to communicate with this machine. The device uses QUARC program version 2.6.

Policy and value iteration algorithms, excluding the stochastic gradient descent method, are built using MATLAB R2017a and Simulink. Interleaved Q-learning and stochastic gradient descent based algorithms are built with Anaconda 4.6 and Python 3.5 using PyTorch, SciPy and NumPy libraries. Windows 10 operating system was used.

3.1.1 Linear system: Quanser QUBE™-Servo 2 and a disk load

Linear control is implemented in a Quanser QUBE™- Servo 2 system with an inertia disk attachment (see Figure 8). In this study, the system model is assumed unknown but for

simulation purposes it is derived in Appendix A. This system is otherwise linear, but the control is constrained between $\pm 10\,V$ [1]. The system is marginally stable.

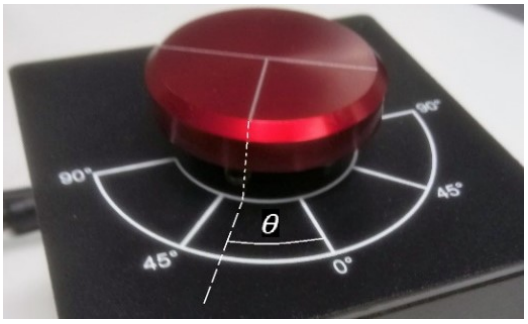

Figure 8. *Quanser QUBE™-Servo 2 with an inertia disk attachment*

The measured angular position θ is marked in Figure 8, the sign of the angular position depends on how the simulator is built. In the linear system simulators, it was chosen positive clockwise.

3.1.2 Nonlinear system: Quanser QUBE™-Servo 2 with an inverted pendulum

The nonlinear system is a Quanser QUBE™-Servo 2 system with an inverted pendulum attachment. Its mathematical model is shown in Appendix B. The model is nonlinear, nonaffine and unstable and it is used to generate data for simulation purposes only.

Figure 9 shows a sketch of the top view and side view of the system. The rotary arm angle θ (see also Figure 8) and the pendulum link angle β are marked in Figure 9. Both of these angles were chosen positive counter clockwise.

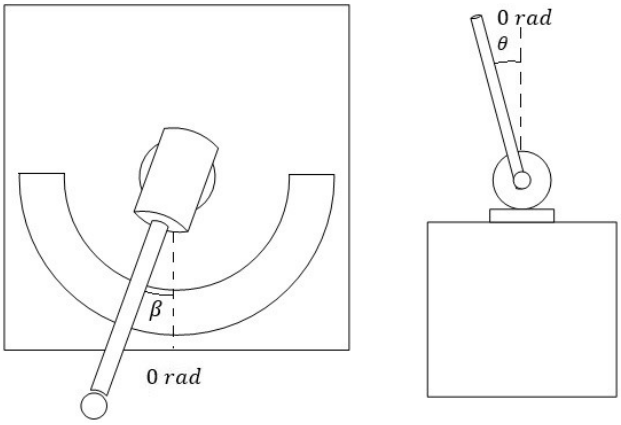

Figure 9. *Top (left) and side view of the Quanser QUBE™-Servo 2 with an inverted pendulum attachment*

The pendulum will be balanced in an upright position while the rotational angle can follow a set trajectory as in [1][29]. The pendulum is rotated upwards manually or by swing-up

control, which is given in more detail in Appendix B. The balancing control is implemented model-free and enabled when the pendulum is near the upright position.

3.1.3 Real-time system resolution and quantized measurements

The Quanser QUBE™- Servo 2 system has features such as limited measurement resolution, measurement noise and saturated control input [1]. The angles θ and β are measured as encoder counts. The system generates 2048 counts per revolution. One revolution in radians is $2\pi\, rad$. To measure radians, the counts are multiplied by a factor ${}^{2\pi}/_{2048}\ rad/counts$. Therefore, the resolution is either $0.17\,°$ or $0.0031\, rad$ [1].

The measurement resolution leads to quantized output. In [18][45], the quantized output y_k^q and control u_{qk}^q are defined as

$$\begin{cases} y_k^q = q(y_k) = \Delta \cdot \left(\left\lfloor {}^{y_k}/_{\Delta} \right\rfloor + {}^{1}/_{2}\right) \\ u_{qk}^q = q(u_{qk}) = \Delta \cdot \left(\left\lfloor {}^{u_{qk}}/_{\Delta} \right\rfloor + {}^{1}/_{2}\right) \end{cases}, \tag{3.1}$$

where $\lfloor y_k/\Delta \rfloor$ and $\lfloor u_{qk}/\Delta \rfloor$ are floor functions of y_k/Δ and u_{qk}/Δ and $\Delta = 0.0031\, rad$. Control u_{qk} is defined by the equation set

$$\begin{cases} u_{qk} = -(T_{uu})^{-1}[T_{u\bar{u}} \quad T_{u\bar{y}}]\bar{x}_{qk} \\ \bar{x}_{qk} = \left[\bar{u}_{qk}^T \quad \bar{y}_{qk}^T\right]^T \\ \bar{u}_{qk} = [\, u_{q(k-1)} \quad u_{q(k-2)} \quad \cdots \quad u_{q(k-n)}]^T \\ \bar{y}_{qk} = \left[\, y_{k-1}^q \quad y_{k-2}^q \quad \cdots \quad y_{k-n}^q\right]^T \end{cases} \tag{3.2}$$

Different types of Kalman filters estimate real states as in [39]. These filters are model-based and therefore cannot be applied into this system. Model-free filters in [28] filter noise online and offline, but their performance is not as good as model-based estimators.

New Bellman equation can be formed using (3.1) and (3.2) with (2.11). With the methods in [18][45] this new Bellman equation converges to the same value as the original Bellman equation in (2.11). However, these methods are finite-horizon algorithms. The algorithm in [18] needs the model of the original system or the optimal target weight must be known. Reference [5] models a new Bellman equation to consider some system disturbances, but little research is found on measurement disturbances.

3.2 Q-learning implementation for linear systems

Policy and value iteration algorithms with least squares, recursive least squares and stochastic gradient descent value updates are implemented using the Figure 7 implemen-

tation steps. Interleaved Q-learning is used only with full state measurements. Simulations are done with the model using both full state and output measurements, but the real-time algorithms are mainly implemented using the output measurements, since full state measurements are not available.

3.2.1 Off-policy policy and value iteration with simulated data

Off-policy policy and value iteration methods are implemented using least squares value updates from Figure 4 and Chapters 2.1.6 and 2.1.7. One article [19] proves that this method of Q-learning is less sensitive to the exploration noise selection than the more traditional approach.

The model from Appendix A is used to generate data. The optimal control is learned offline with a batch of data that is collected with a behaviour policy and added exploration noise to satisfy the PE condition. The batch size M needs to be large enough to have enough information about the system.

```
if PI == true
   PHI= phi-gamma.*phi_next; Y = diag(x.'*Q*x+u.'*R*u) ;
else
   PHI= phi; Y = diag(x.'*Q*x+u.'*R*u+gamma.*Wold.'*phi_next);
end
W = (PHI*PHI.')\PHI*Y;
```

Program 1. *Policy and value iteration value update using LS*

```
if PI == true
    y= x_.'*Q*x_+ u.'*R*u;
    fun= @(Smat) y-z.'*Smat*z+gamma*z_next.'*Smat*z_next;
    S=lsqnonlin(fun,S);
else
    KSI= x_.'*Q*x_+ u.'*R*u+gamma*z_next.'*S*z_next;
    Y=KSI*z.'/(z*z.'); PHI=z.';
    S= (PHI.'*PHI)\PHI.'*Y;
end
```

Program 2. *Policy and value iteration value update using LS2*

The same batch is used repeatedly to update the value. The least squares value update methods are shown in Program 1 and Program 2 and. Program 1 is based on equations (2.52) and (2.54) and Program 2 on $(2.55) - (2.68)$. Policy iteration on Program 2 is implemented using least squares and nonlinear least squares in MATLAB using `nonlinlsq`-tool. The learned optimal control can be inserted into the simulated system after convergence.

3.2.2 Off-policy interleaved Q-learning with simulated data

Interleaved Q-learning implementation follows the steps in Figure 6 and Chapter 2.2.1. First, the training data is collected and then the optimal control is learned offline with a set of data that is collected with a stabilizing behaviour policy.

The data is collected near the control region in episodes. Each episode starts at the same initial position. The generated random noise seed is changed for each episode. Data is collected until all the chosen amount of episodes are performed and each time k is separated into their own batch.

3.2.3 On-policy PI and VI in a simulated environment

Policy and value iteration algorithms with recursive least squares and stochastic gradient descent value updates are implemented as on-policy methods. These algorithms are built using the architecture shown in Figure 2 and they follow the steps in Figure 3 and Chapters 2.1.6, 2.1.8 and 2.1.9.

```
if PI == true
   Y= (x.'*Q*x+u.'*R*u);
   PHI = phi-gamma*phi_next;
else
   Y= (x.'*Q*x+u.'*R*u)+gamma*Wold.'*phi_next;
   PHI = phi;
end
L= 1/lambda.*P*PHI/(1+1/lambda.*PHI.'*P*PHI); %gain
Wnew = Woldrls +L*(Y-PHI.'*Woldrls); %weight
P =1/lambda.*(eye(length(W))-L*PHI.')*P; %covariance update
```

Program 3. *PI and VI Recursive least squares value update with MATLAB*

```
def SGD(gamma,phi,phi_next,r,Ws,Wold,alpha,PI,batch):
   for i in range(batch):
         Q_est = Ws.T @ phi[:,[i]];
         if (PI==true):
               Qn_est = gamma * Ws.T @ phi_next[:,[i]];
               Q = r[:,[i]] + Qn_est;
               Ws = Ws – alpha * \
               (phi[:,[i]] - gamma*phi_next[:,[i]]) @ (Q_est-Q)
         else:
               Qn_est= gamma * Wold.T @ phi_next[:,[i]];
               Q = r[:,[i]] + Qn_est;
               Ws = Ws – alpha * (phi[:,[i]]) @ (Q_est-Q);
   return Ws
```

Program 4. *PI and VI SGD value update with Python*

The recursive least squares value update is given in Program 3. It uses the equations $(2.70) - (2.72)$.The weight $\widehat{W}_{j+1,i}$ is named as `Woldrls`, $\widehat{W}_{j+1,i+1}$ as `Wnew` and the weight

$\widehat{W}_j$ as `Wold`. Value update for SGD is given in Program 4 and it uses equations $(2.74) - (2.78)$. The weight `Ws` is the weight $\widehat{W}_{j+1,i}$ that is being updated by SGD and `Wold` is the last converged weight $\widehat{W}_j$. Stochastic gradient descent algorithm is built on Python, but since the gradient in equation (2.77) is calculated with matrix equations, the real-time implementation is built in MATLAB. It was tested, but not proven here, that this makes the algorithm faster as the conversions between MATLAB vectors and Python arrays are omitted.

3.2.4 From theoretical environment to real-time environment

From here forward, the simulators use $n = n_x = 2$. At first, the data is simulated in MATLAB using the model in Appendix A without disturbances. This is called original simulator. The simulator in Figure 10 considers the measurement resolution and disturbances and it is called modified simulator. The measurement resolution is implemented with a quantizer block and the input control is limited between $\pm 10\,V$ with a saturation block. Optional disturbances are also added to the system.

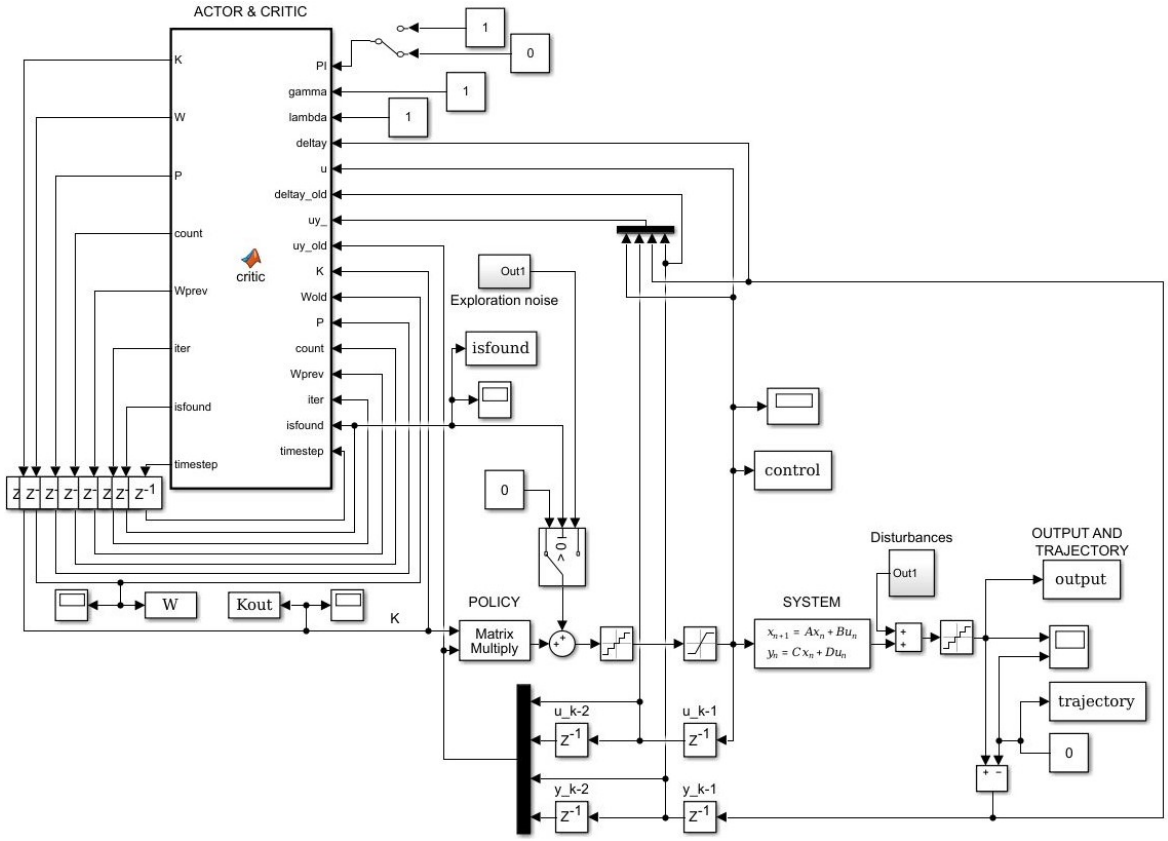

Figure 10. *PI and VI RLS modified simulator including saturation, quantizer and disturbances around the simulated model*

Figure 11. *`ACTOR & CRITIC` block for on-policy LS*

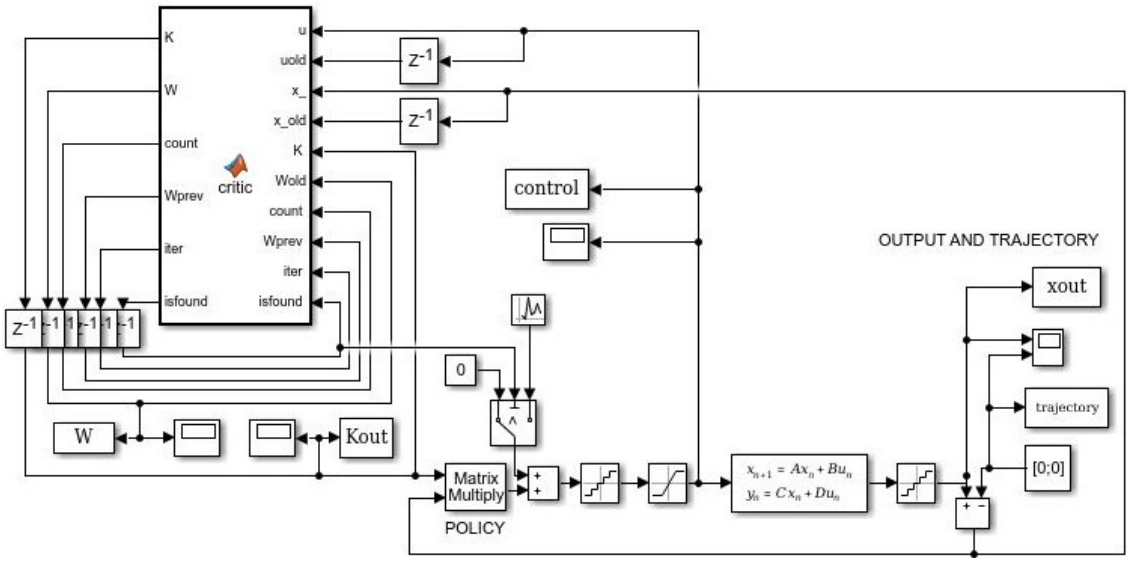

Figure 12. *PI and VI SGD full-state simulator*

Figure 11 shows an `ACTOR & CRITIC` block addition that inserted into Figure 10 when on-policy LS methods are used instead of RLS methods. Figure 12 show the simulator when full state measurements are used with on-policy SGD methods.

3.2.5 Off-policy methods with real data

A batch of data is collected using a simulator that communicates with the real-time machine. The simulator with angular position measurements is shown in Figure 13. Figure 14 shows the simulator with an additional estimated velocity. The first simulator is used for off-policy PI and VI algorithms and latter is used for interleaved Q-learning.

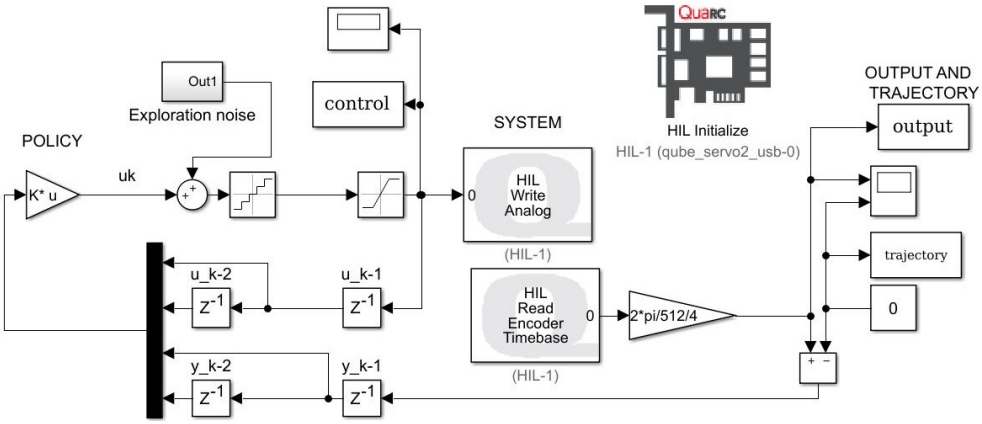

Figure 13. *Simulator to collect data from the real-time system*

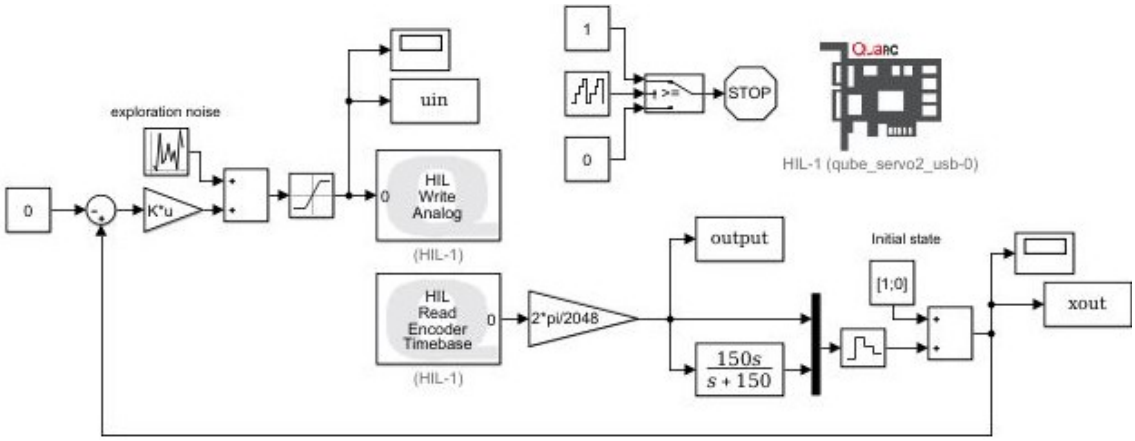

Figure 14. *Simulator to collect full state data from the real-time system*

Least square algorithms use data from only one experiment, whereas interleaved Q-learning uses n_{sets} learning episodes. The noise seed is changed between runs.

3.2.6 On-policy methods in a real-time environment

The simulator from Figure 10 is updated so that the system model, the quantizer and the noise blocks are replaced by the real-time system as shown in Figure 15. Similarly the real-time system addition in Figure 16 replaces parts of the simulator in Figure 12. The output is multiplied by a factor of $2\pi/2048$ to change the counts into radians as in [1].

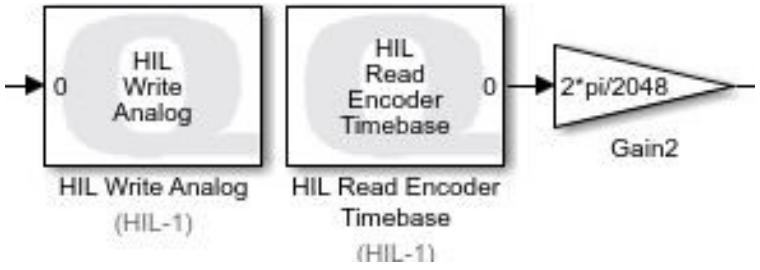

Figure 15. *Real-time system addition to measure the angular position*

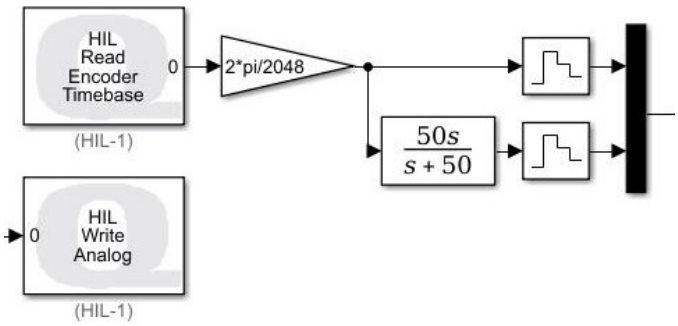

Figure 16. *Real-time system addition when the velocity is estimated with a high-pass filter*

The control noise and sample time are chosen within the limits of the system. The measured angle was chosen positive clockwise. For opposite sign, the input control and output measurement need an additional minus sign.

3.3 Q-learning implementation for nonlinear system

Interleaved Q-learning for nonlinear systems uses only full state data. The algorithm is built in Anaconda Python, but the data is imported to MATLAB for simulation purposes. Since the system model in Appendix B is nonaffine and the Q-learning algorithm used is developed for affine systems, the data is collected near the upright position, where the system can be assumed affine or linear.

3.3.1 Off-policy interleaved Q-learning with simulated data

Nonlinear system measurements are first collected with the simulator in Figure 17 using stabilizing behaviour policy. The algorithm follows the procedure in Figure 6 and Chapter 2.2.1. Several learning episodes are repeated to collect enough data. Exploration noise needs to be chosen so that the system does not become unbalanced and that the system is still within the affine control region. For each episode, the noise seed is changed. Collected data is saved and exported from MATLAB to Python using SciPy as is shown in Program 5.

```
import scipy.io as sp
data = sp.loadmat('sourcefilename.mat')
X = torch.tensor(data['xdata'], dtype = torch.float32);
U = torch.tensor(data['udata'], dtype = torch.float32);
# lines of code removed
datastorage = {};
Weight = Weighttensor.numpy() #PyTorch tensor into NumPy array
datastorage['Weightname'] = Weight;
sp.savemat('targetfilename.mat', datastorage)
```

Program 5. *Exporting the learned weight matrices between MATLAB and Python*

The same dataset is reused throughout the learning phase. Larger data batch results in better learning according to article [32]. However, in this study, it was empirically seen that too large batch size might cause RAM-memory issues.

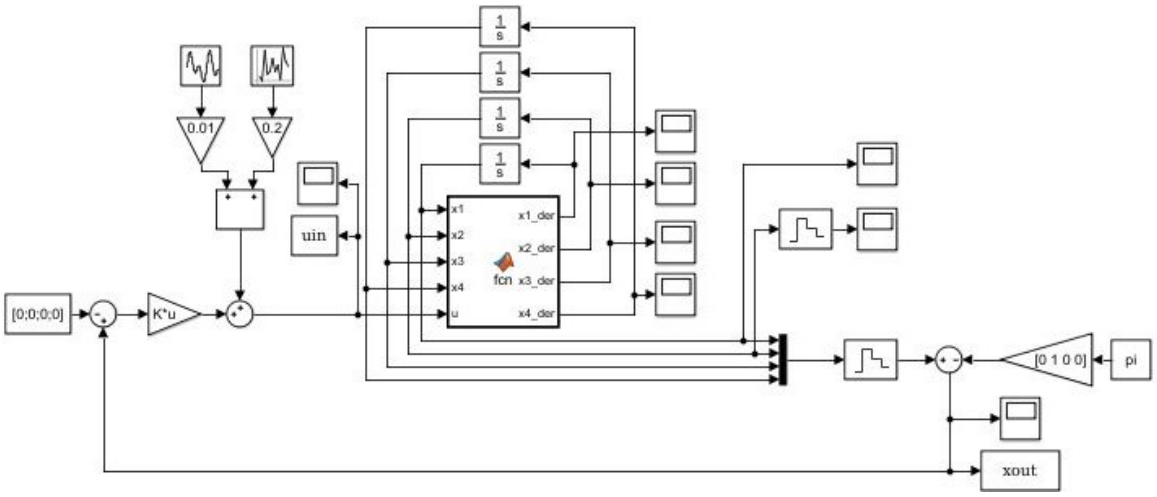

Figure 17. *Nonlinear system model for simulating data*

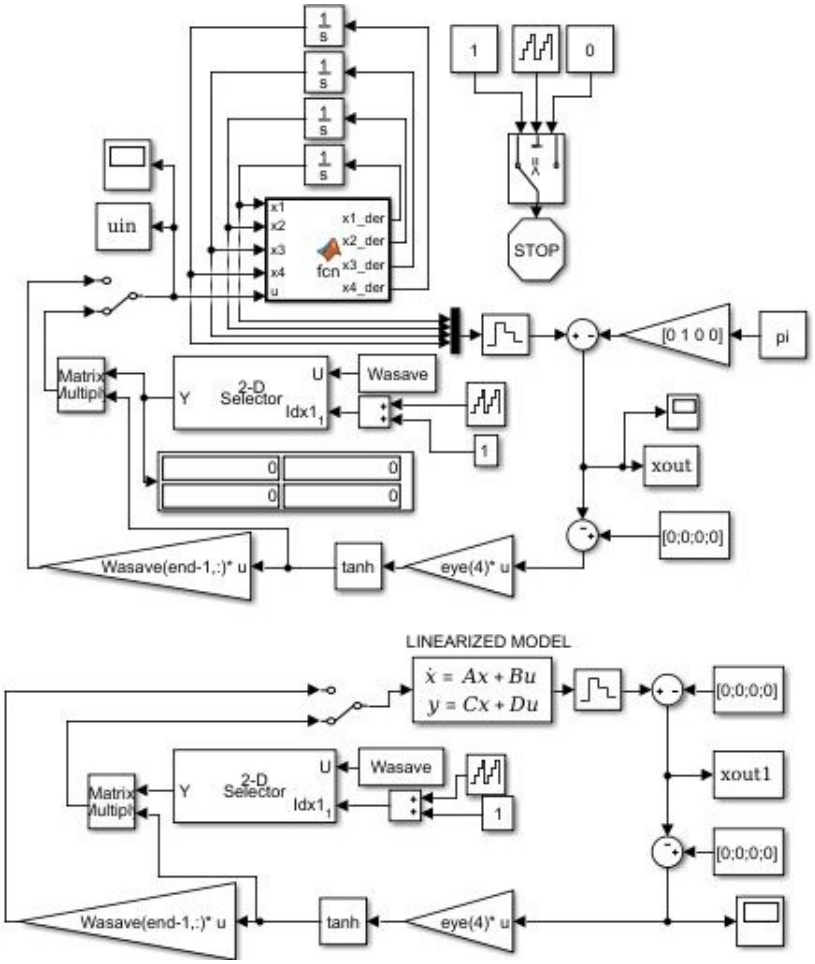

Figure 18. *Nonlinear system model (above) and linearized system model (below) with feedback control loop using the learned policy*

The algorithm follows the steps in Figure 6. PyTorch is used for gradient calculation.

3.3.2 Off-policy interleaved Q-learning with real-time data

The real-time environment replaces the simulated environment in Chapter 3.3.1. The pendulum is rotated upwards manually between each learning episode and the noise seed is changed each run.

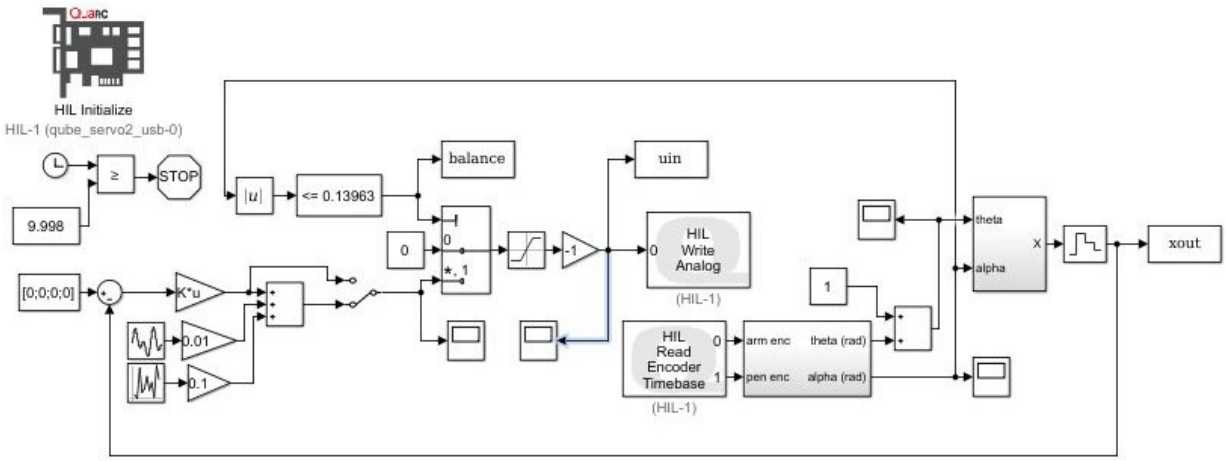

Figure 19. *Nonlinear real-time system simulator, modified from [1]*

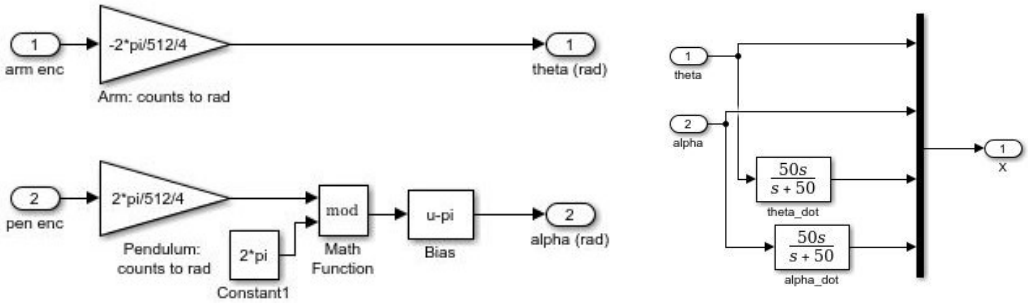

Figure 20. *The two subplots from Figure 19, modified from [1]*

Figure 19 and Figure 20 show the nonlinear simulator that was used to collect data from the nonlinear real-time system. Collected data is processed so that each episode starts from the same initial position. The episodes are saved in a large matrix containing all episodes under each other. This matrix is used in the interleaved Q-learning algorithm given in Figure 6.

4. LINEAR SYSTEM Q-LEARNING RESULTS

Several different Q-learning methods (see Table 1) found in literature were first implemented for a simulated version of the Quanser QUBE™ -Servo 2 experiment with inertia disk attachment using a model from Appendix A. Then, the theoretical models were modified to generate data that resembles the real-time system. Lastly, the theoretical model was replaced by the real-time system.

Table 1. *Algorithm name abbreviations used in the results*

Name	Meaning
LS	Least squares weight matrix value update
LS2	Least squares kernel matrix value update
PI	Policy iteration algorithm
RLS	Recursive least squares value update
LQR ref.	Results with the known model
SGD	Stochastic Gradient descent value update
VI	Value iteration algorithm

Q-learning uses no information about the system dynamics excluding the output or state feedback measurements and control actions. Off-policy methods for the linear system were PI and VI algorithms with LS value updates and interleaved Q-learning and on-policy methods were PI and VI algorithms with LS, RLS and SGD value updates.

4.1 Theoretical results using full state measurements

The initial parameters and constant variables for the theoretical policy and value iteration algorithms with full state measurements are given in Table 2. PI and VI SGD algorithms are initialized with different initial gain and kernel matrix, as the one used with the other algorithms does not lead to convergence with PI SGD.

Table 2. *Theoretical PI and VI algorithms initial values and constant variables*

Parameter	Chosen values
LQR parameters	$Q = 20I,\ R = 1, \gamma = 1$
Kernel matrix	$S_0 = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$ or $S_{0,SGD} = \begin{bmatrix} 1214.3 & 16.4 & 27.5 \\ 16.4 & 2.2 & 2.7 \\ 27.5 & 2.7 & 7.6 \end{bmatrix}$
Gain	$\hat{K}_0 = [-0.5 \quad -0.5]$ or $\hat{K}_{0,sgd} = [-3.62 \quad -0.36]$
Batch size	$2n_z(n_z + 1)/2$, where $n_z = 3$
Exploration noise	Gaussian random noise
Sample time	$dt = 0.01\ s$

4.1.1 Theoretical results using off-policy methods with full state measurements

The convergence of the PI and VI algorithms was very similar with both LS methods (LS and LS2), so the following figures only show the results of the LS method. Convergence limit was chosen as $\varepsilon_j = 10^{-2}$ and the values and constants from Table 2 were used. Noise was chosen in MATLAB as `noise = randn(1,N)`.

Figure 21 shows how the learned model-free gain $\widehat{K}_j$ and kernel matrix $\hat{S}_j$ compare to the optimal model-based solution calculated with (2.4) and (2.14). The difference is calculated as the norm of the error between the learned value and the optimal value.

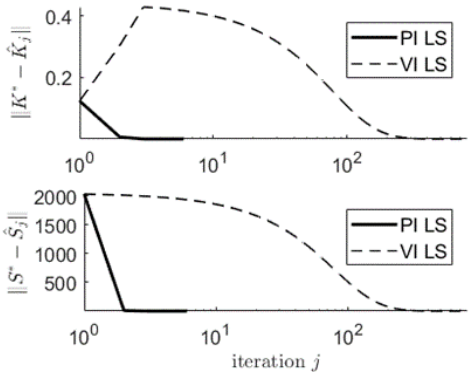

Figure 21. *The PI and VI LS gain $\widehat{K}_j$ and kernel matrix $\hat{S}_j$ compared to the optimal gain K^* and the kernel matrix S^**

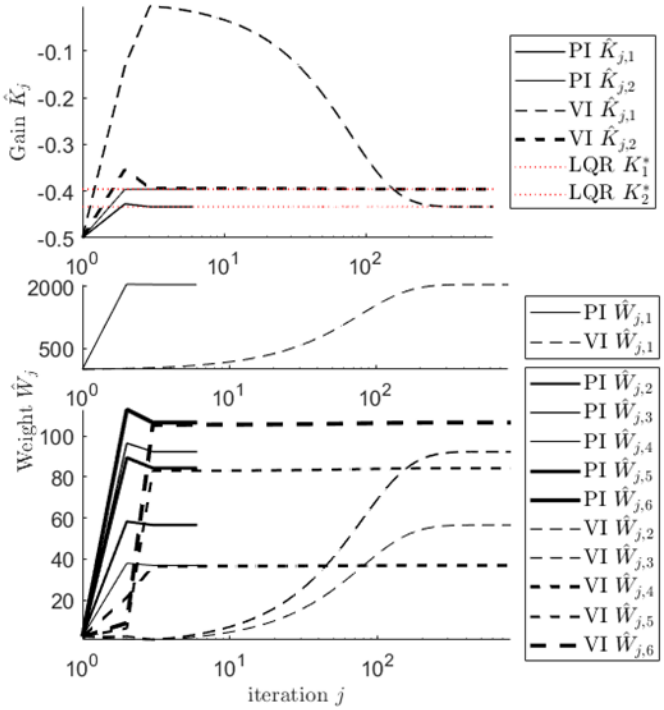

Figure 22. *Evolution of the PI and VI LS gain $\widehat{K}_j$ and weight $\widehat{W}_j$*

Figure 22 shows the evolution of the gain $\widehat{K}_j$ and weight $\widehat{W}_j$. The final gain $\widehat{K}_j$, kernel matrix $\hat{S}_j$ and the number of iterations j (policy updates) are shown on Table 3. The first row is the optimal model-based solution and the following rows are the learned gains $\widehat{K}_j$ from each off-policy PI and VI algorithms. The learned linear control policy is $u_k = \widehat{K}_j x_k$.

Table 3. *The off-policy PI and VI results with full state measurements*

Algorithm	**Full state gain $\widehat{K}_j$**	**Kernel matrix $\widehat{S}_j$**	**Iterations j**
LQR ref.	$[-0.4334 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	-
PI off-policy			
LS2	$[-0.4333 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	3 updates
LS	$[-0.4334 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	3 updates
VI off-policy			
LS2	$[-0.4333 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	451 updates
LS	$[-0.4333 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	451 updates

Interleaved Q-learning actor, critic and model network structures were chosen as 2-2-1, 3-5-1 and 3-10-2, where the first element is the number of network inputs $n_z = 3$ or $n_x = 2$, the second is the number of hidden layer neurons n_{nw} in each network and the third is the number of network outputs as in [19][42]. The initial weights were selected for the actor, the critic and the model network as $W_{a,0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $W_{c,0} = J_c$ and $W_{m,0} = J_m$, where $J_c \in \mathbb{R}^{n_c x 1}$ and $J_m \in \mathbb{R}^{n_m x n_x}$ are matrices of ones. The learning rates were selected as $\alpha_a = \alpha_c = 0.001$ and $\alpha_m = 0.01$. Convergence limit was $\varepsilon_j = 10^{-1}$ and the algorithm is stopped when $\left\| \widehat{W}_{c,j}(k) - \widehat{W}_{c,j}(k+1) \right\| \leq 2 \cdot 10^{-3}$. The hidden layer weights v_a , v_c and v_m were chosen randomly and kept constant as in [19]. The weights v_a, v_c and v_m were chosen with Program 6, where nz= n_z, va = v_a, vm = v_m, vc= v_c, nx = n_x, nha = n_a, nhm = n_m and nhc = n_c.

```
import torch
va = torch.eye(nx);
vc = 0.1*torch.randn(nz,nhc);
vm = 0.1*torch.randn(nz,nhm);
```

Program 6. *The hidden layer weights for Interleaved Q-learning*

Figure 23 shows the evolution of the weights during the offline interleaved Q-learning. The final policy is $\hat{u}_{\infty,k} = \widehat{W}_{a,\infty}^{T}\sigma(v_a^T x_k)$, where $\widehat{W}_{a,\infty}$ is the learned actor network weight. The final actor weights were $W_{a,\infty} = [-0.1341 \quad -0.0652]$. Interleaved Q-learning used a batch of data with $n_{sets} = 100$ learning episodes with $N = 2000$ time steps.

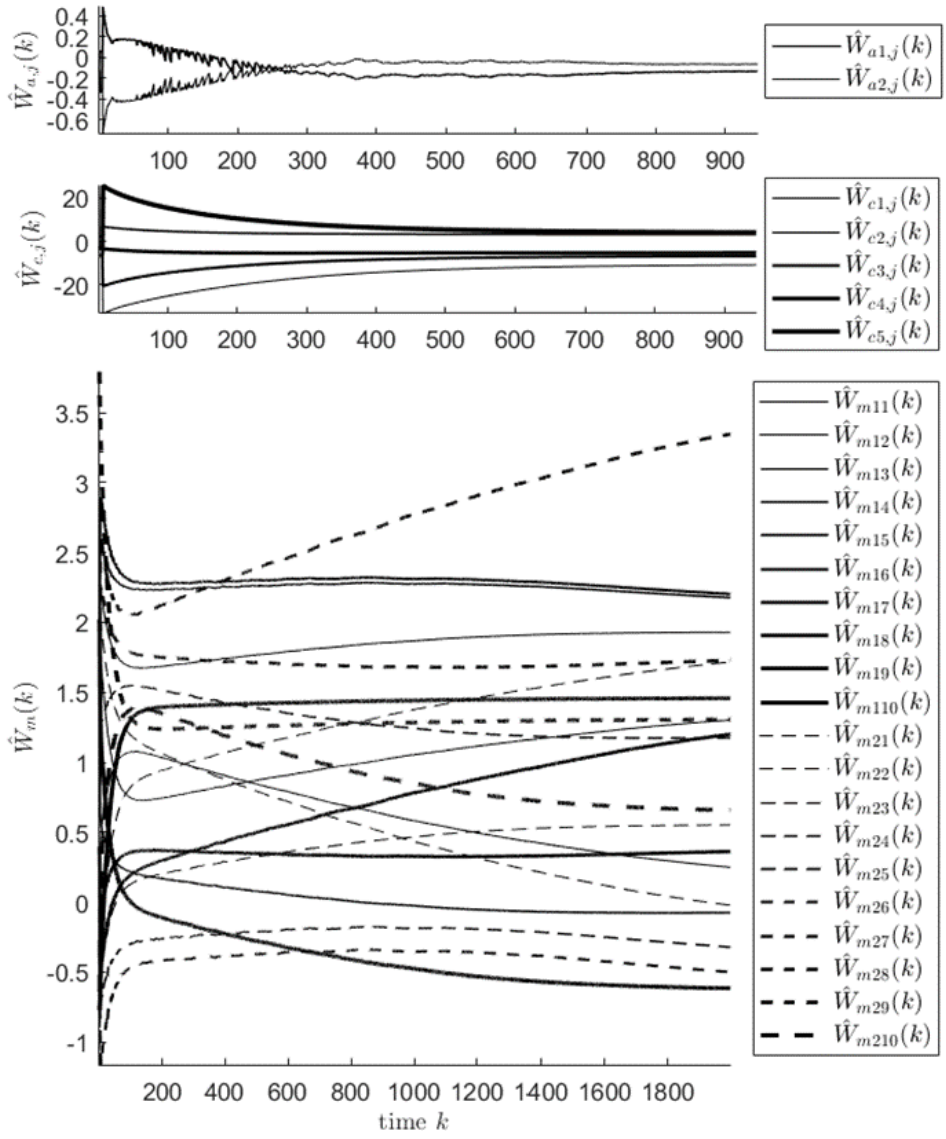

Figure 23. *Evolution of the interleaved Q-learning full state weights $\widehat{W}_{a,j}(k)$, $\widehat{W}_{c,j}(k)$ and $\widehat{W}_m(k)$ when the learning is successful*

The optimal policy from the interleaved Q-learning algorithm cannot be compared directly to the control gains in Table 3. Instead, the learned control policy of the PI LS from Table 3 and the learned interleaved Q-learning control policy were used with the theoretical system model from Appendix A. The state trajectories when the initial state was $\mathrm{x}_0 = [1 \quad 1]^{\mathrm{T}}$ are shown in Figure 24 (left). The learned policy was also tested outside the training region. Figure 24 shows the state trajectories also with an initial position $\mathrm{x}_0 = [2 \quad -2]^{\mathrm{T}}$ and a goal position $\mathrm{x}_\infty = [2 \quad 0]^{\mathrm{T}}$.

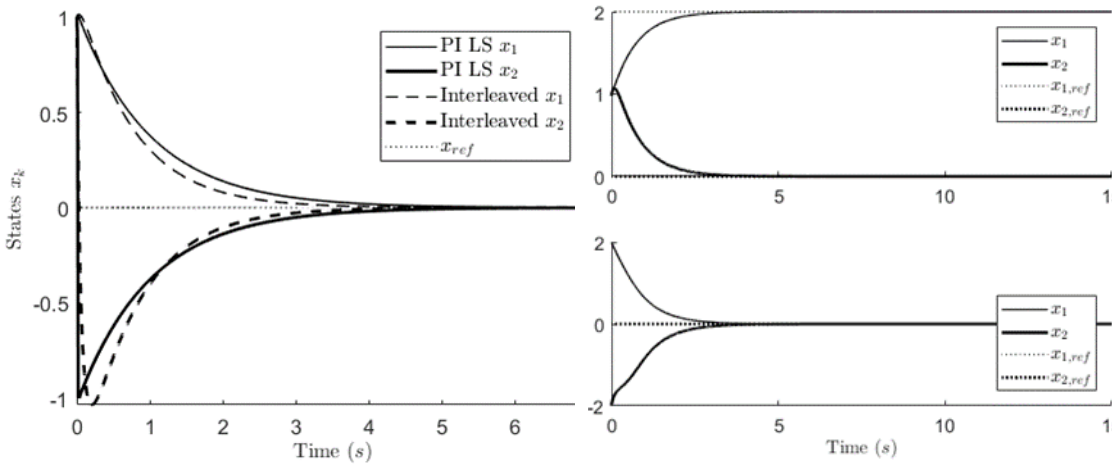

Figure 24. *Using the learned interleaved Q-learning control policy when the initial state is $x_0 = [1 \quad 1]^T$ (left) or when the final state is $x_\infty = [2 \quad 0]^T$ (right above) or when initial state is $x_0 = [2 \quad -2]^T$ (right below)*

4.1.2 Theoretical on-policy results with full state measurements

On-policy methods were the PI and VI algorithms with LS, RLS and SGD value updates. The initial values and variables from Table 2 were used. The RLS discounting factor was chosen as $\lambda = 1$, convergence limits were $\varepsilon_i = 10^{-7}$and $\varepsilon_j = 10^{-9}$ and the covariance matrix $P_0 = 100I$. Learning rates were chosen for the VI SGD as $\alpha_{sgd} = 0.001$ and for the PI SGD as $\alpha_{sgd} = 0.01$, convergence limits were $\varepsilon_i = 10^{-2}$ and $\varepsilon_j = 10^{-7}$. For RLS methods the exploration noise was chosen in MATLAB as a Gaussian random noise as `noise = 2*randn(1,N)`. For SGD, the noise was chosen in Python as a uniform random noise as `noise= 2.5*np.random.rand(N+1,1).` Both platforms had the maximum time step set as `N=100000.`

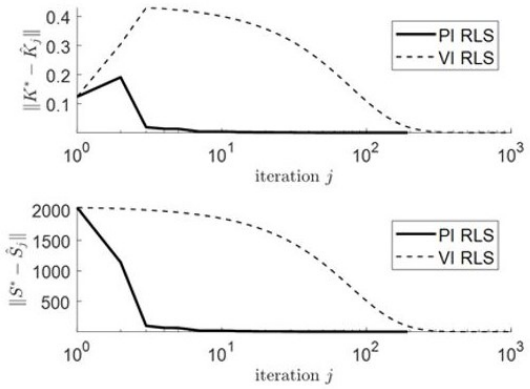

Figure 25. *The PI and VI RLS full state gain $\hat{K}_j$ and kernel matrix $\hat{S}_j$ compared to the optimal gain $\mathrm{K}^*$and the kernel matrix S^**

Figure 25 and Figure 28 show the norm of the difference between the optimal gain K^* and the learned gain $\widehat{K}_j$ besides the norm of the difference between the optimal kernel matrix S^* and the learned kernel matrix $\hat{S}_j$ when using RLS and SGD algorithms. Figure 27 and Figure 29 show the evolution of the gain $\widehat{K}_j$ and weight $\widehat{W}_j$ when using these algorithms. The control and states during the RLS PI and VI is introduced in Figure 26 the initial state is $x_0 = [0 \quad 0]^T$.

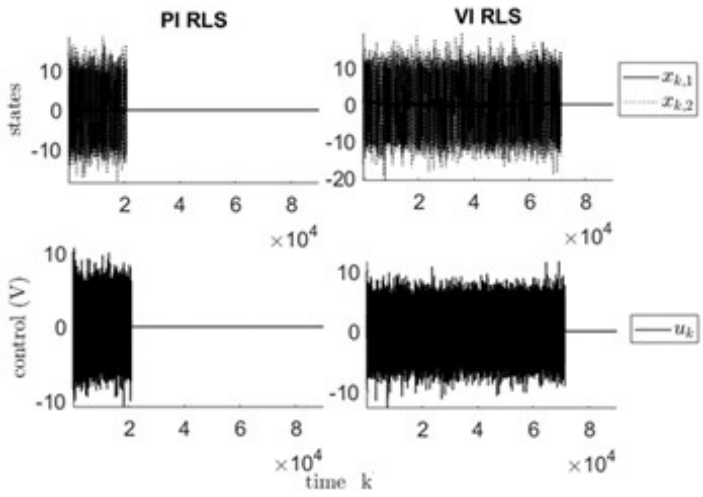

Figure 26. *Control and state signals of the PI and VI RLS algorithms when the initial state is* $x_0 = [0 \quad 0]^T$

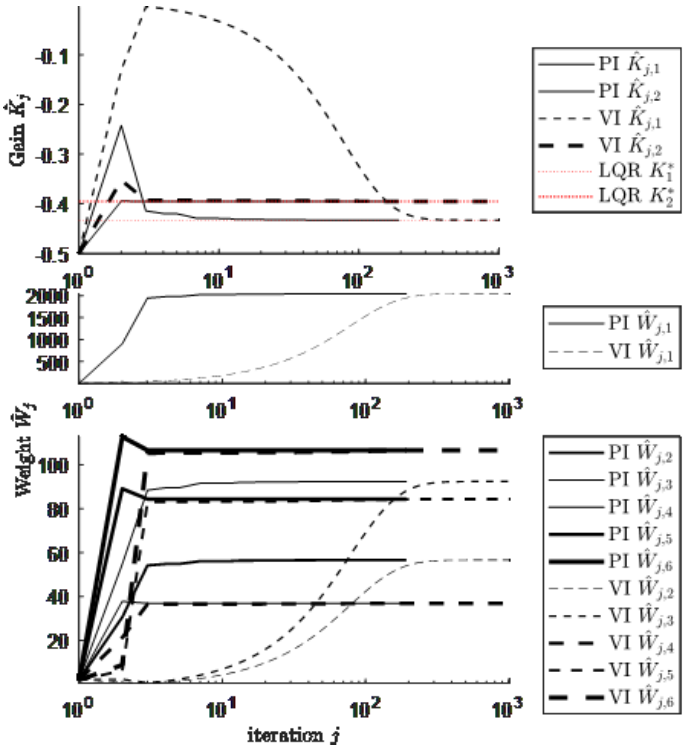

Figure 27. *Evolution of the PI and VI RLS full state gain* $\widehat{K}_j$ *and weight* $\widehat{W}_j$

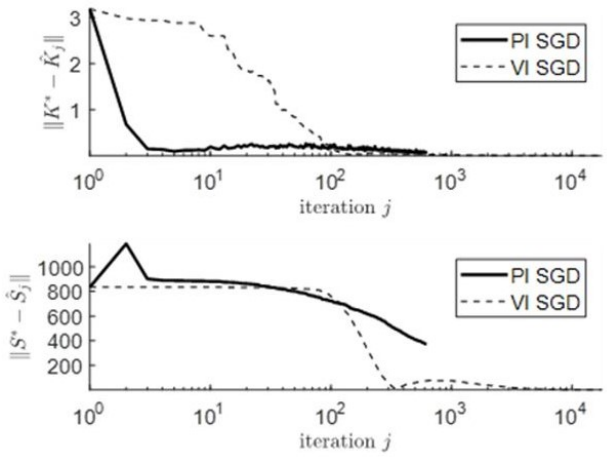

Figure 28. *The PI and VI SGD learned full state gain $\hat{K}_j$ compared to the optimal gain K^**

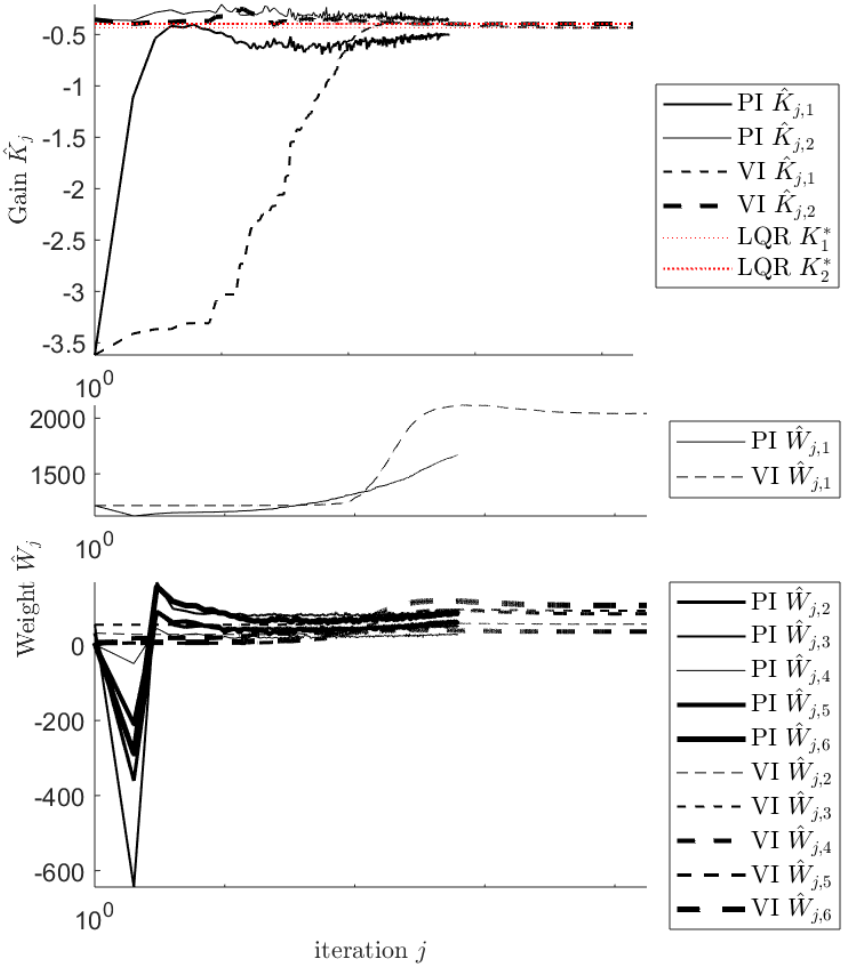

Figure 29. *Evolution of the PI and VI SGD full state gain $\hat{K}_j$ and weight $\hat{W}_j$*

Convergence with on-policy LS methods was similar with off-policy LS methods in Chapter 4.1.2 and the detailed figures are not shown here. The convergence limits, exploration noise and other variables were selected the same as with RLS.

Table 4. *The on-policy PI and VI results with full state measurements*

Algorithm	Full state gain $\widehat{K}_j$	Kernel matrix $\widehat{S}_j$	Iterations j
LQR ref.	$[-0.4334 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	-
PI on-policy			
RLS	$[-0.4334 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	191
SGD	$[-0.4902 \quad -0.3595]$	$\begin{bmatrix} 1703.3 & 24.8 & 42.9 \\ 24.8 & 31.8 & 31.5 \\ 42.9 & 31.5 & 87.5 \end{bmatrix}$	661(at maximum time k)
LS	$[-0.4333 \quad -3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	7
VI on-policy			
RLS	$[-0.4334 \quad -0.3957]$	$\begin{bmatrix} 2040.1 & 28.3 & 46.2 \\ 28.3 & 36.9 & 42.1 \\ 46.2 & 42.1 & 106.5 \end{bmatrix}$	1007
SGD	$[-0.4773 \quad -3964]$	$\begin{bmatrix} 2048.9 & 28.3 & 46.2 \\ 28.3 & 36.8 & 42.3 \\ 46.2 & 42.3 & 106.5 \end{bmatrix}$	10693
LS	$[-0.4333 \quad -3957]$	$\begin{bmatrix} 2039.9 & 28.3 & 46.1 \\ 28.3 & 36.9 & 42.1 \\ 46.1 & 42.1 & 106.5 \end{bmatrix}$	499

The learned gains $\widehat{K}_j$ and kernel matrices $\hat{S}_j$ are shown in Table 4. Only PI SGD does not converge within the set limit of $N = 100000$ time steps.

4.2 Theoretical results using output measurements

Policy and value iteration algorithms were implemented using output feedback as was mentioned in the implementation steps in Figure 7. For these algorithms the observability index was chosen as $n = n_x = 2$. The initial values and constant variables are given in Table 5.

Table 5. *The theoretical PI and VI algorithms' initial values and constant variables*

Parameter	Chosen values
LQR parameters	$Q = 20, R = 1, \gamma = 1$
Kernel matrix	$T = \begin{bmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{bmatrix}$
Gain	$\widehat{K}_0 = [-0.5 \quad -0.5 \quad -0.5 \quad -0.5]$
Batch size	$3 \cdot n_{\bar{z}}(n_{\bar{z}} + 1)/2$, where $n_{\bar{z}} = 5$
Exploration noise	Gaussian random noise
Sample time	$dt = 0.01\ s$

4.2.1 Theoretical results using off-policy methods with output feedback measurements

PI and VI LS and LS2 were used with the output feedback measurements. The initial values and constant variables were given in Table 5 and the exploration noise and convergence limits were the same in Chapter 4.1.1. Figure 30 shows the gain $\widehat{K}_j$ compared to the optimal. Figure 31 and Figure 32 show the evolution of the weight $\widehat{W}_j$ and gain $\widehat{K}_j$ when a set of data was used to learn the policy using the least squares value update.

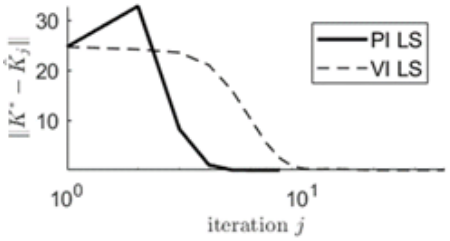

Figure 30. *The PI and VI LS output feedback gain $\widehat{K}_j$ compared to the optimal output feedback gain K^**

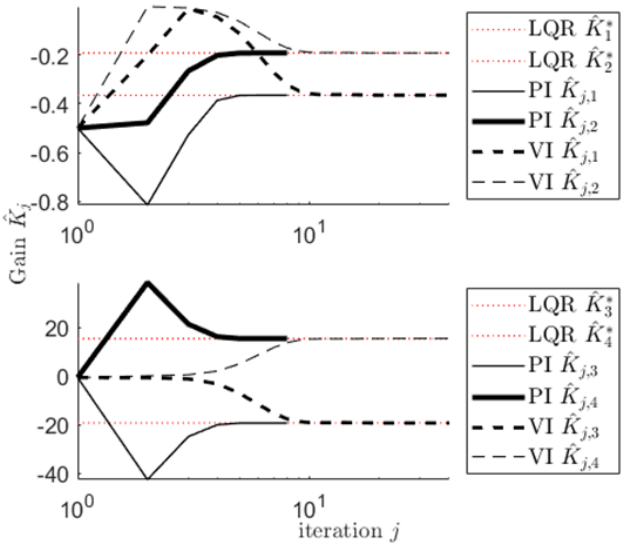

Figure 31. *The evolution of the PI and VI LS output feedback gain $\widehat{K}_j$*

Table 6. *Off-policy PI and VI results with output feedback*

Algorithm	Output feedback gain $\widehat{K}_j$	Iterations j
LQR ref.	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$	-
PI off-policy		
LS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$	6
LS2	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$	6
VI off-policy		
LS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5364]$	38
LS2	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5364]$	38

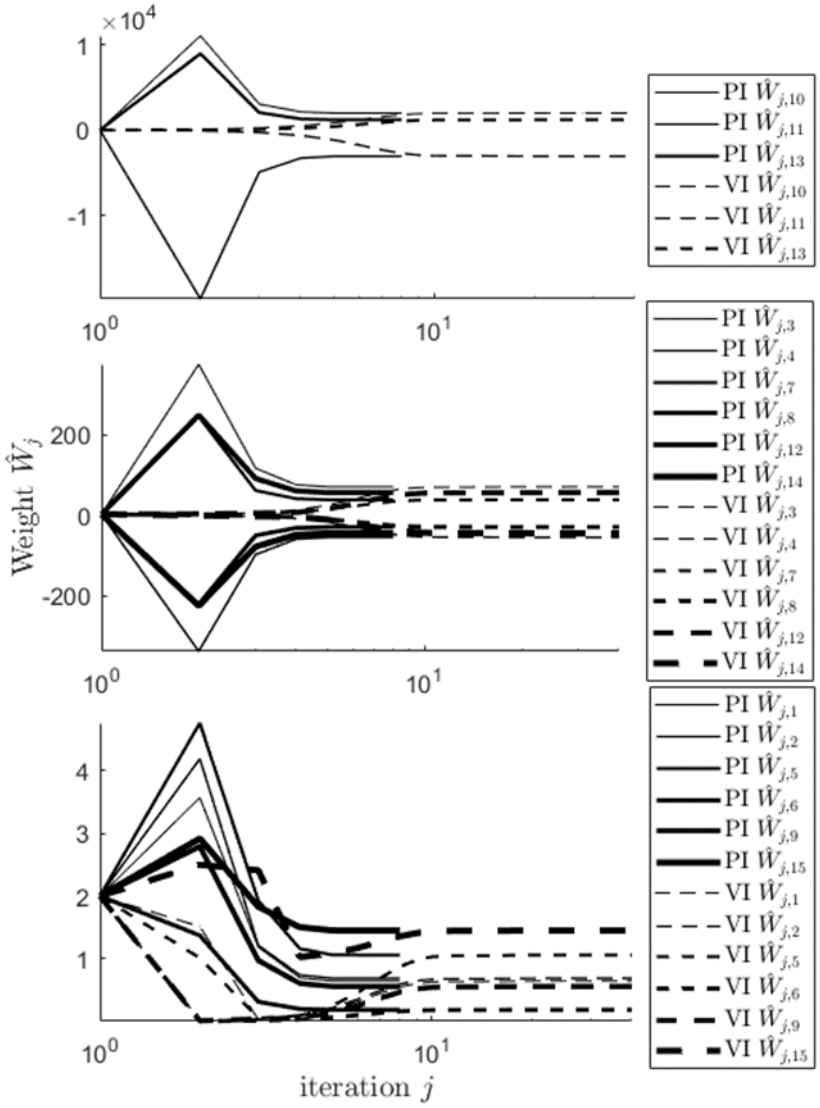

Figure 32. *Evolution of the PI and VI LS output feedback weights $\widehat{W}_j$*

The learned gains are shown in Table 6. The first row is the optimal gain. Due to similar results, detailed results of PI and VI LS2 are omitted.

4.2.2 Theoretical on-policy results with output feedback measurements

The PI and VI algorithms with RLS, SGD and LS value updates were used with output feedback measurements. The convergence limits for the PI and VI RLS and LS algorithms are the same as in Chapter 4.1.2. The initial values and constant variables were given in Table 5. The exploration noise for the RLS and LS algorithms was selected in MATLAB as `noise = 2*randn(1,N).` Figure 33 shows the gain $\widehat{K}_j$ compared to the optimal. Figure 34 and Figure 35 show the evolution of the gain $\widehat{K}_j$ and weight $\widehat{W}_j$. Converged weights are listed in Table 7.

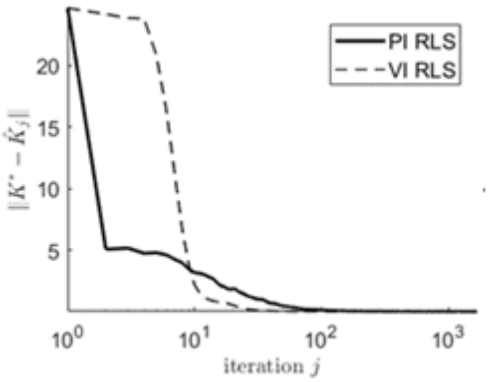

Figure 33. *Evolution of the PI and VI RLS output feedback gain $\hat{K}_j$ compared to the optimal gain K^**

Table 7. *On-policy PI and VI results using output feedback $\lambda = 1$*

Algorithm	Output feedback gain $\hat{K}_j$	Iterations *j*
LQR ref.	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$	-
PI on-policy		
RLS	$[-0.3654 \quad -0.1922 \quad -19.2503 \quad 15.5370]$	1676
LS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$	23
VI on-policy		
RLS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$	1675
LS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$	54

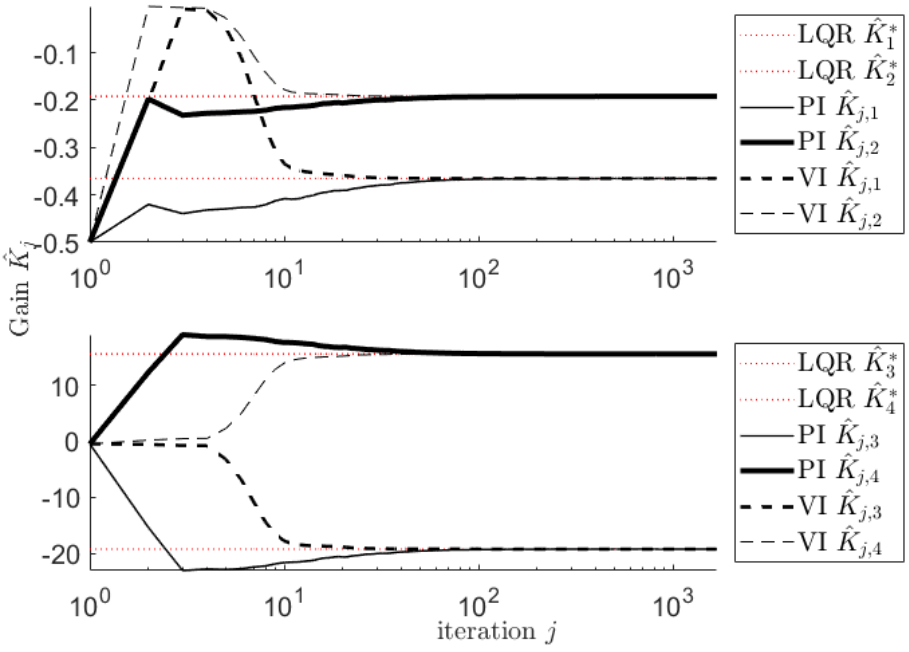

Figure 34. *Evolution of the PI and VI RLS output feedback gain $\hat{K}_j$*

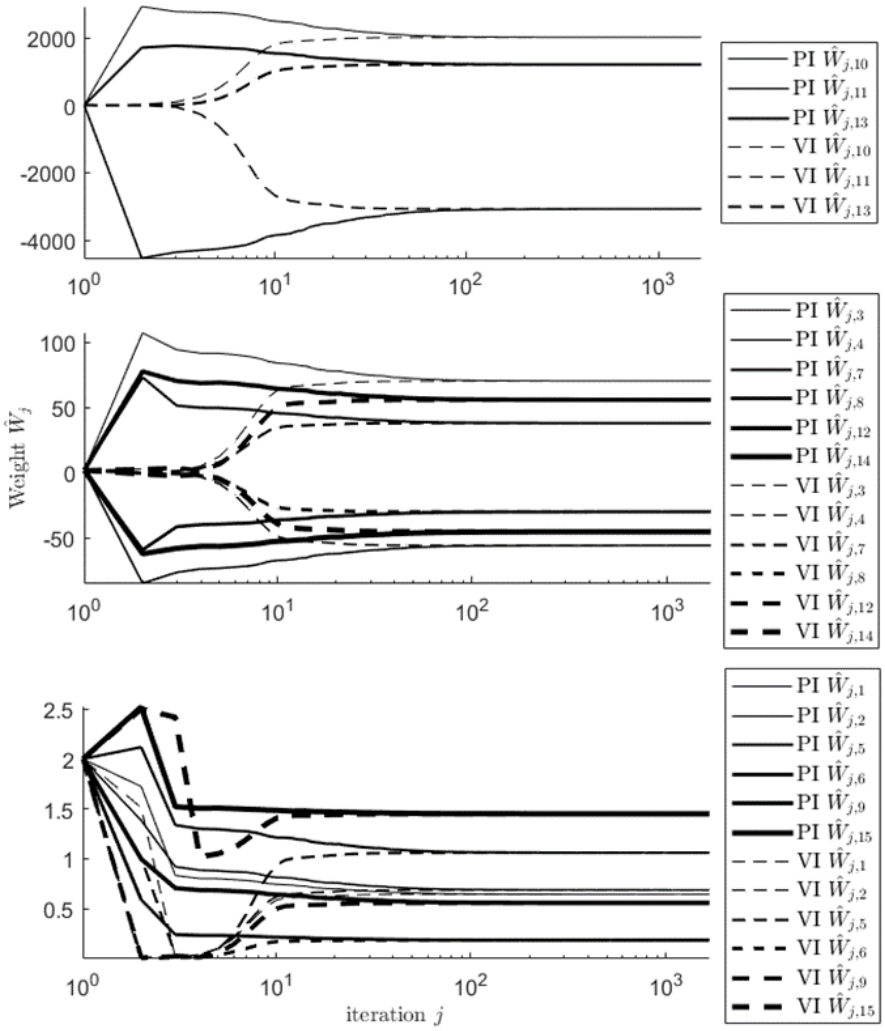

Figure 35. *Evolution of the PI and VI RLS output feedback weight $\widehat{W}_j$*

The author did not find an exploration noise nor a learning rate that makes the PI and VI SGD output feedback algorithms converge to the optimal solutions. This might be due to lack of independence between the samples [7].

4.3 On-policy output feedback results with modified simulator

Figure 36 demonstrates a situation where the simulated actual output is presented with the quantized output. This data is simulated using the system model from Appendix A, and constant gain $K = [-0.5 \quad -0.5 \quad -0.5 \quad -0.5]$ and an initial position at $y_k = 0\,rad$. A uniform random noise is added to the signal and the noise amplitudes are given in Figure 36. Figure 36 shows that the measurement error caused by the quantizer is proportionally larger with smaller sample time and smaller noise amplitude. Quantization causes error in the measurements that can lead to errors in the Bellman equation as was explained in Chapter 3.1.3. Due to results in Figure 36, the sample time of the theoretical algorithms $dt = 0.002\ s$ is increased to $dt = 0.01\ s$ for the real-time control.

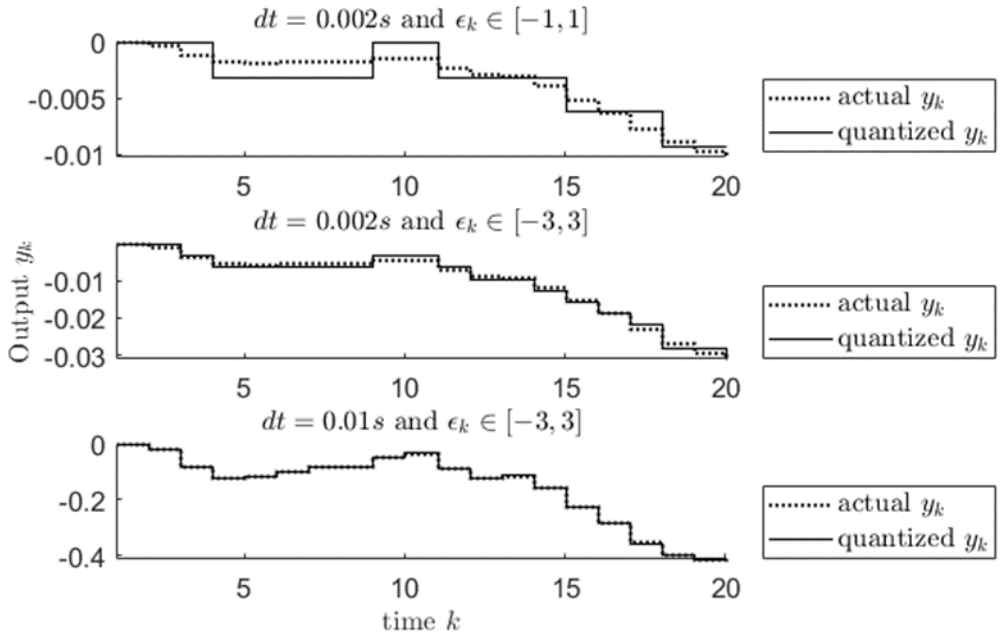

Figure 36. *The effect of the quantizer with different noises and sample times*

The modified simulator uses output feedback measurements. A stabilizing control is calculated using the theoretical model with $Q = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$ and $R = 1$ as finding a stabilizing gain experimentally was challenging. Table 8 shows the initializing parameters and values used in the modified simulator.

Table 8. *The real-time PI and VI algorithms initial values and constant variables*

Parameter	**Chosen values**
LQR parameters	$Q_y = 1, R = 1, \gamma = 1$
Kernel matrix	$T = \begin{bmatrix} 0.346 & 0.180 & 17.641 & -14.519 & 0.304 \\ 0.180 & 0.094 & 9.264 & -7.575 & 0.157 \\ 17.641 & 9.264 & 924.819 & -748.720 & 15.195 \\ -14.519 & -7.575 & -748.720 & 612.172 & -12.673 \\ 0.304 & 0.157 & 15.195 & -12.673 & 1.273 \end{bmatrix}$
Gain	$K = [-0.2391 \quad -0.1232 \quad -11.9369 \quad 9.9550]$
Batch size	$15 \cdot n_z(n_z + 1)/2$, where $n_z = 5$
Sample time	$dt = 0.01\ s$

Control noise is chosen as a uniform random noise in $[-5\ V, 5\ V]$. Convergence limits are now $\varepsilon_i = 10^{-5}$and $\varepsilon_j = 10^{-5}$ and control input is saturated within $[-7\ V, 7\ V]$.

4.3.1 Simulator with measurement quantization and input saturation

First, with the given initial values and variables in Table 8 and RLS discounting factor $\lambda = 1$, the PI and VI RLS simulator in Figure 10 was run without disturbances or discontinuities (original simulator). These simulations were then repeated using the simulator

with measurement quantization and input saturation but without other disturbances (modified simulator). The evolution of the gain $\widehat{\mathrm{K}}_{\mathrm{j}}$ and the weight $\widehat{\mathrm{W}}_{\mathrm{j}}$ during these two simulations are shown in Figure 37 and Figure 38. Figure 39 shows the learning within time from $k = 19990$ to $k = 20000$. The learning is noisy, and it does not converge.

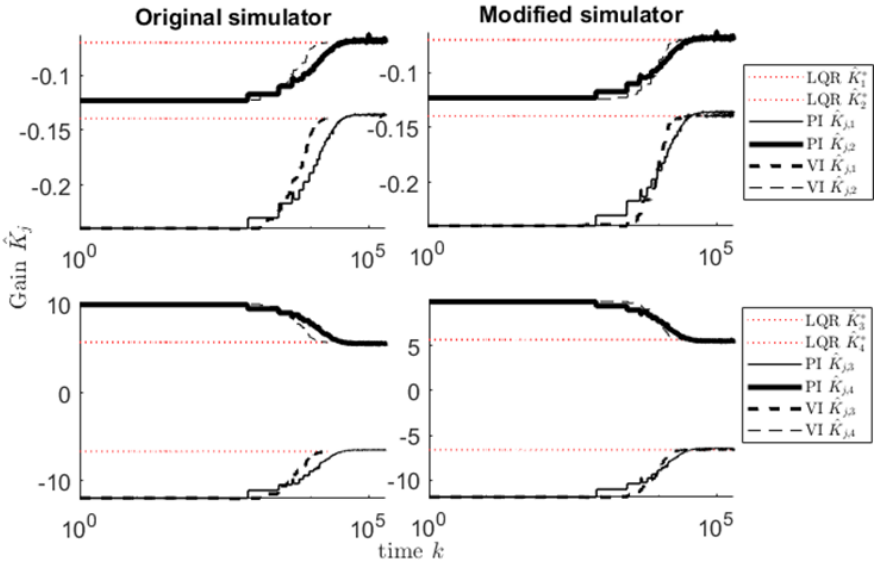

Figure 37. *Evolution of the PI and VI RLS gain $\widehat{K}_j$ with the original and the modified simulator*

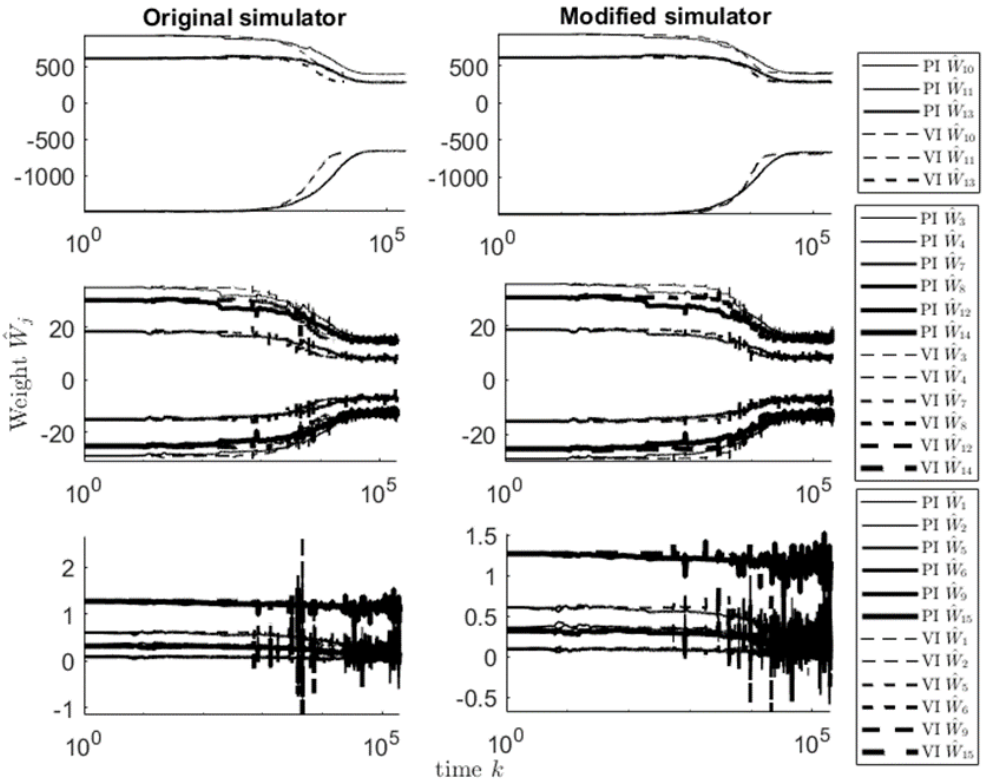

Figure 38. *Evolution of the PI and VI RLS weight $\widehat{W}_j$ with the original and the modified simulator*

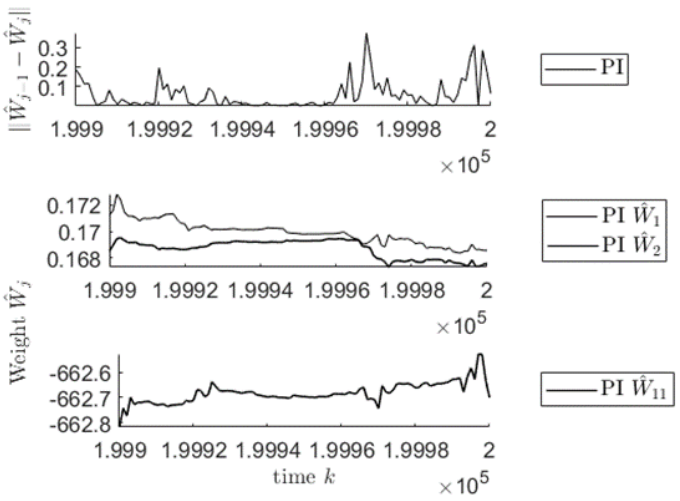

Figure 39. *The convergence of the weight $\hat{W}_j$ with the PI RLS modified simulator within time steps from $k = 19990$ to $k = 20000$*

Figure 40 and Figure 41 show on-policy PI and VI LS simulations without disturbances (original simulator) and with quantization and input saturation (modified simulator). Simulators were run using the PI and VI RLS simulator in Figure 10 with the LS addition from Figure 11 and values from Table 8. The learned gains are given in Table 9, where the first row is the optimal control calculated using the known model and equation (2.29).

Table 9. *The on-policy PI and VI modified simulator results*

Algorithm	Output feedback gain $\hat{K}_j$
LQR ref.	$[-0.1395 \quad -0.0706 \quad -6.6362 \quad 5.7039]$
PI on-policy	
RLS	$[-0.1358 \quad -0.0683 \quad -6.4770 \quad 5.5655]$
LS	$[-0.1339 \quad -0.0688 \quad -6.3723 \quad 5.4696]$
VI on-policy	
RLS	$[-0.1387 \quad -0.0708 \quad -6.6064 \quad 5.6820]$
LS	$[-0.1410 \quad -0.0701 \quad -6.6806 \quad 5.7462]$

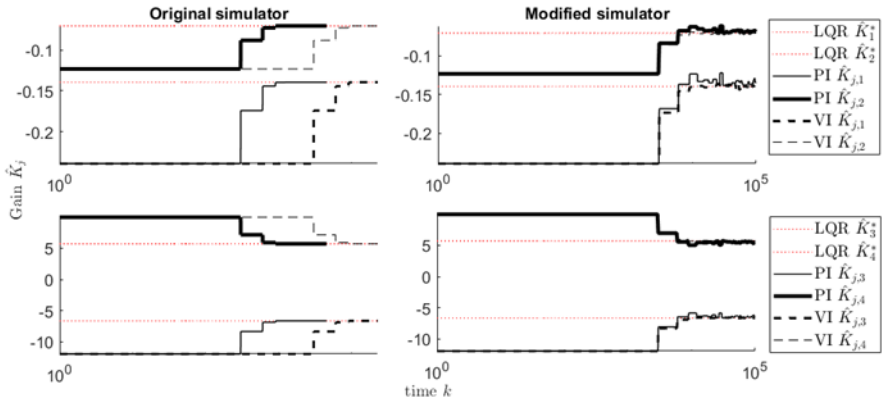

Figure 40. *The on-policy PI and VI LS with the original and the modified simulator gain $\hat{K}_j$*

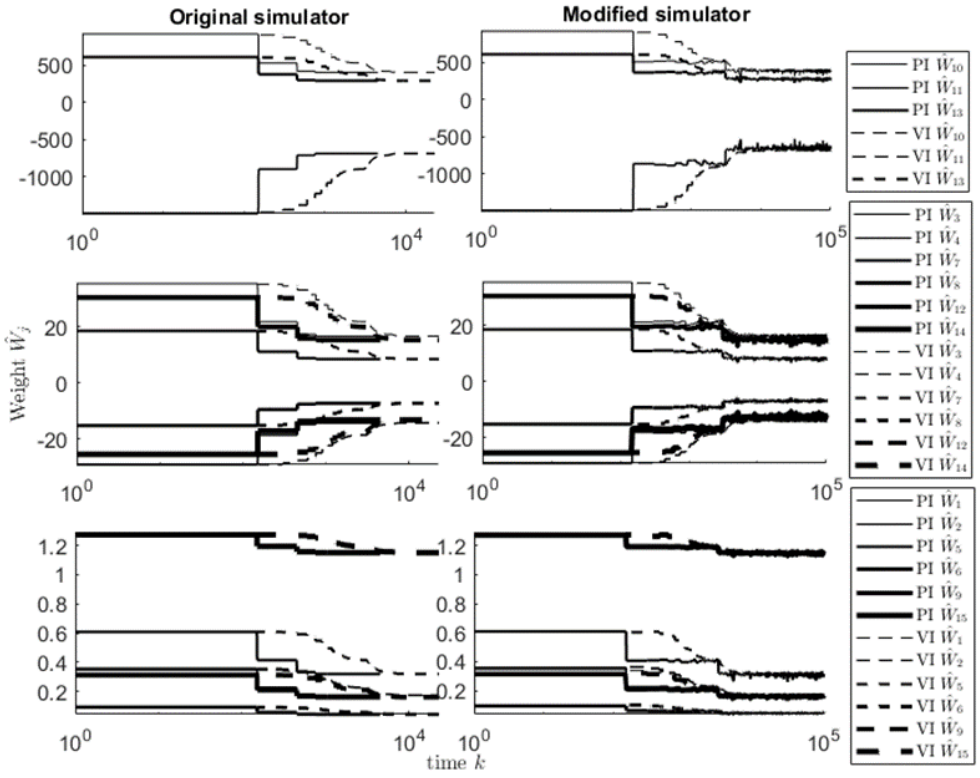

Figure 41. *Evolution of the on-policy PI and VI LS weight $\hat{W}_j$ with the original and the modified simulator*

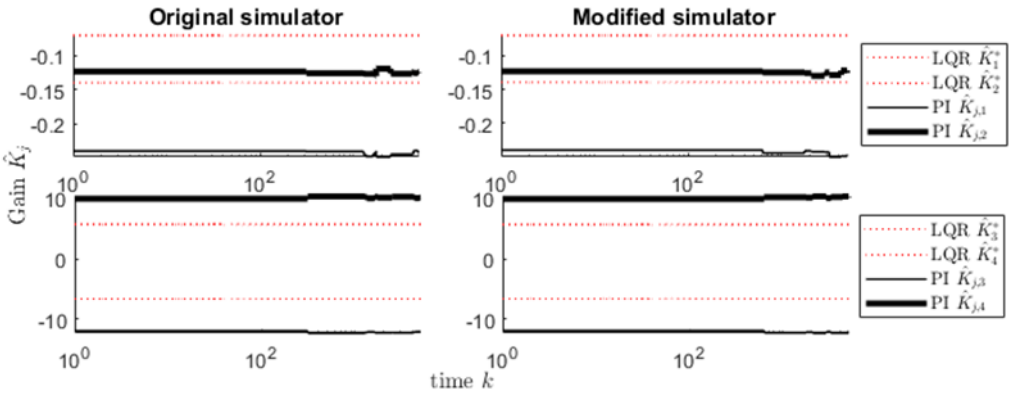

Figure 42. *Small exploration noise in the original and the modified simulator*

Figure 42 shows that selecting exploration noise with a small variance might not lead to proper learning with neither the original model nor the modified simulator.

4.3.2 Simulation with quantization and measurement noise

The results of Chapter 4.3.1 are repeated with added measurement noise. Table 8 values were used with the modified simulator. Three different variances were tested and the results with PI and VI RLS and LS online are shown in Figure 43 and Figure 44. Measurement noise is assumed Gaussian white noise.

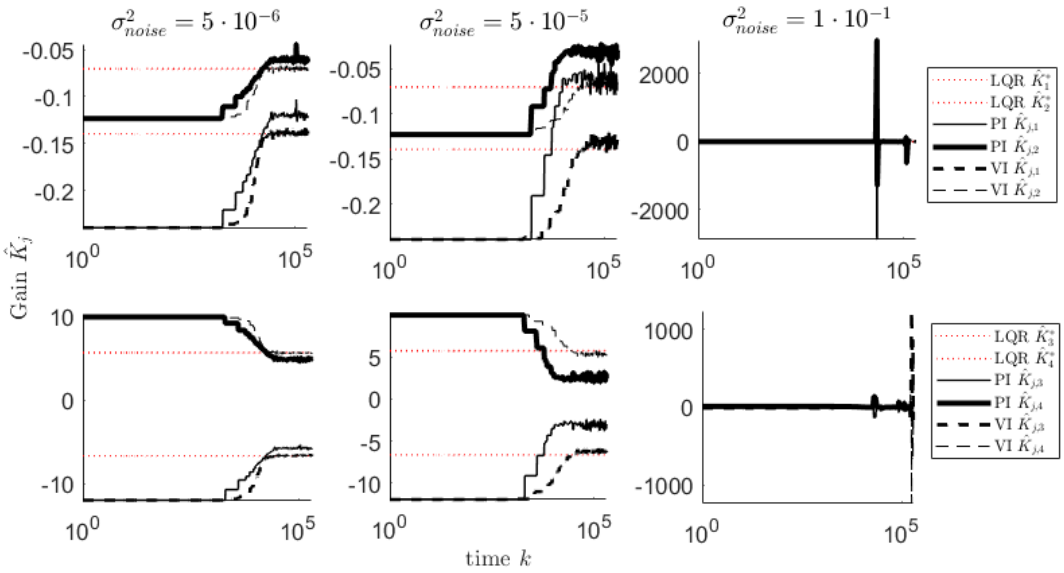

Figure 43. *Evolution of the gain $\hat{K}_j$ with the PI and VI RLS when measurement noise is present*

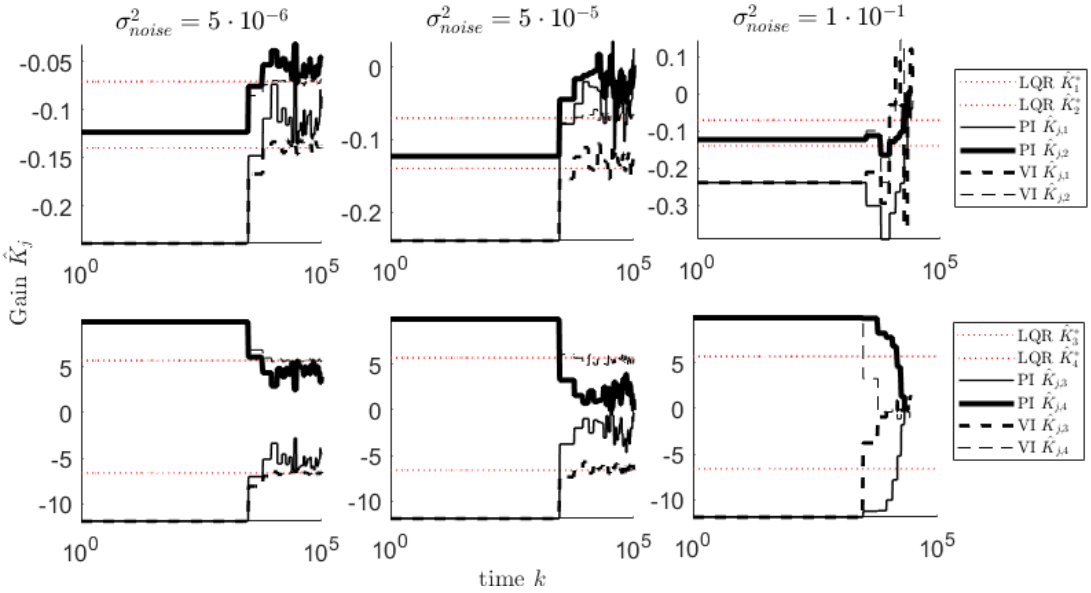

Figure 44. *Evolution of the gain $\hat{K}_j$ with the on-policy PI and VI LS when measurement noise is present*

As can be seen in Figure 43 and Figure 44, increasing the noise will cause more error in learning. In these examples, value iteration tolerates noise better than policy iteration.

4.3.3 Discounting factor in RLS

Discounting factor was earlier chosen as $\lambda = 1$. Figure 45 shows the PI and VI RLS results if it is chosen as $\lambda = 0.8$. Simulations were run with the convergence limits from Chapter 4.3 and values from Table 8 using the simulator in Figure 10. Measurement noise variance is $\sigma^2_{noise} = 5 \cdot 10^{-5}$.

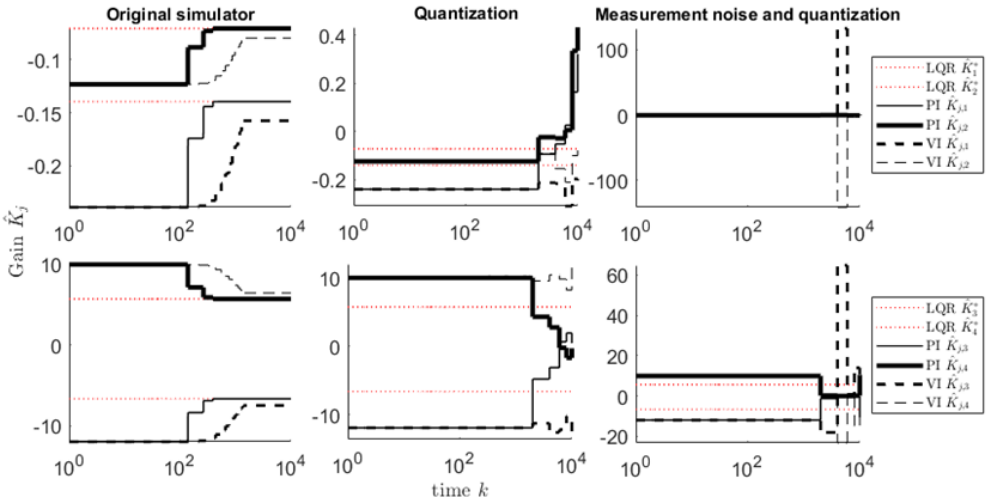

Figure 45. *The original theoretical model, the model with the quantizer and the model with the measurement noise and the quantizer results when* $\lambda = 0.8$

Compared to the original simulator in Figure 37 with a discounting factor $\lambda = 1$, the original simulator in Figure 45 that with the discounting factor $\lambda < 1$ converges faster. Adding quantization and measurement noise causes larger learning error with $\lambda = 1$ in Figure 37 and Figure 43.

4.4 Real-time results

The output feedback algorithms were implemented in the real system using Simulink and the Quanser QUBE™ -Servo 2 experiment with the inertia disk attachment and the initial values from Table 8. Figure 46 shows 3 sets of data collected with the real-time system using the simulator in Figure 10 with the real-time addition in Figure 15. The real-time datasets are presented with a set of data that was simulated without disturbances using the simulator in Figure 10.

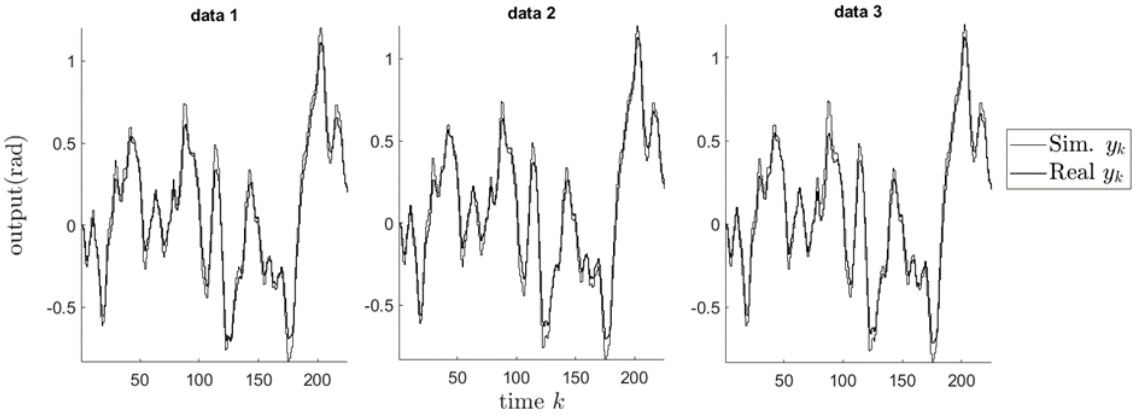

Figure 46. *The real-time control and output signals presented with the simulated control and output signals of 3 different experiments*

The two first datasets are from PI and VI RLS real-time runs that lead to successful policy (later shown in Figure 55) and the last dataset is from an unsuccessful learning trial (in Figure 50). The data is presented for the first 225 time steps, when the policy is not updated yet.

4.4.1 Off-policy results using real-time data

The 3 sets of data in Figure 46 were used with the off-policy PI and VI methods. Table 8 values were used with convergence chosen as $\varepsilon_j = 10^{-2}$.

Table 10. *Off-policy results when using the real-time system data*

Algorithm	Output feedback gain $\hat{K}_j$	Iterations *j*
LQR ref.	$[-0.1395 \quad -0.0706 \quad -6.6362 \quad 5.7039]$	-
PI off-policy		
LS data 1	$[-0.0611 \quad -0.0371 \quad -3.3740 \quad 3.0246]$	34
LS data 2	$[-0.0727 \quad -0.0426 \quad -4.1807 \quad 3.7191]$	27
LS data 3	$[-0.0848 \quad -0.0509 \quad -4.8278 \quad 4.2489]$	12
LS2 data 1	$[-0.0066 \quad -0.0038 \quad -0.3643 \quad 0.3952]$	82
LS2 data 2	$[-0.0033 \quad -0.0018 \quad -0.1836 \quad 0.2390]$	250
LS2 data 3	$[-0.0445 \quad -0.0264 \quad -2.4660 \quad 2.2140]$	40
VI off-policy		
LS data 1	$[-0.1124 \quad -0.0704 \quad -6.6310 \quad 5.8029]$	42
LS data 2	$[-0.1156 \quad -0.0692 \quad -7.0236 \quad 6.1361]$	36
LS data 3	$[-0.1112 \quad -0.0678 \quad -6.5225 \quad 5.6938]$	43
LS2 data 1	$[-0.1091 \quad -0.0665 \quad -6.4905 \quad 5.6637]$	39
LS2 data 2	$[-0.1100 \quad -0.0665 \quad -6.5406 \quad 5.7083]$	39
LS2 data 3	$[-0.1099 \quad -0.0662 \quad -6.2925 \quad 5.4742]$	38

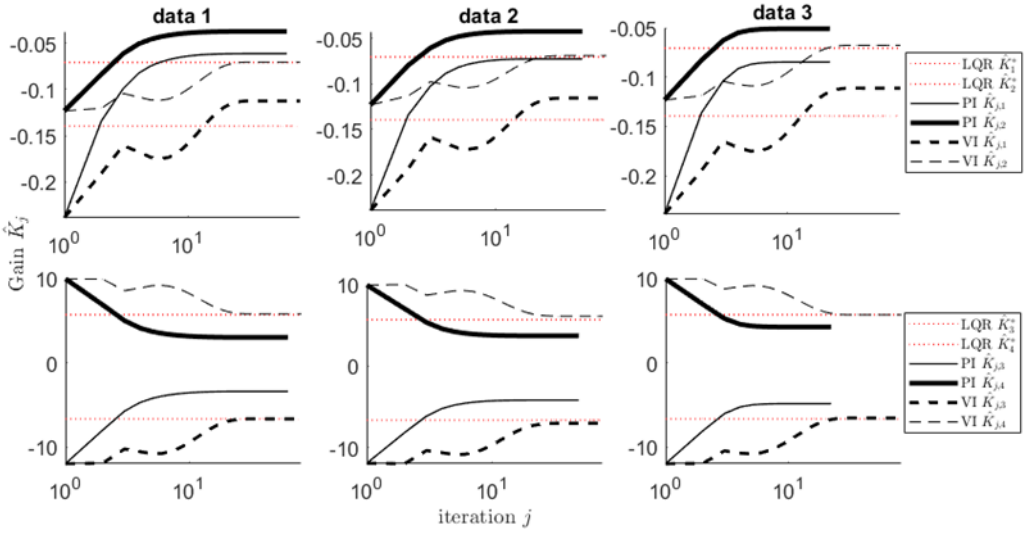

Figure 47. *Evolution of the PI and VI LS gain $\hat{K}_j$ with 3 real datasets*

Figure 47 shows the convergence of the gain $\widehat{\mathrm{K}}_{\mathrm{j}}$ with the different datasets. The exploration noise was a uniform random noise between $[-5\,V, 5\,V]$ and control saturation was limited within $[-7\,V, 7\,V]$. Table 10 presents the learned policies when datasets 1-3 are used with PI and VI with LS and LS2 value updates.

Figure 48 shows the evolution of the network weights $\widehat{\mathrm{W}}_{\mathrm{a,j}}(\mathrm{k})$, $\widehat{\mathrm{W}}_{\mathrm{c,j}}(\mathrm{k})$ and $\widehat{\mathrm{W}}_{\mathrm{m}}(\mathrm{k})$ when interleaved Q-learning was used with a batch of real data with $n_{sets} = 10$ learning episodes and $N = 500$ time steps. The network structures for the model, the actor and the critic network are 3-10-2, 2-2-1 and 3-5-1. The initial weights were given in Chapter 4.1.1. The learning rates were chosen as $\alpha_a = \alpha_c = 0.003$ and $\alpha_m = 0.1$. The exploration noise was chosen as a uniform random noise between $[-0.2\,V, 0.2\,V]$ and the behaviour policy was chosen as $u_k = [-0.3793 \quad -0.3420]x_k$. Larger exploration noise amplitude caused instability with the selected behaviour policy. Data was collected using the simulator in Figure 14.

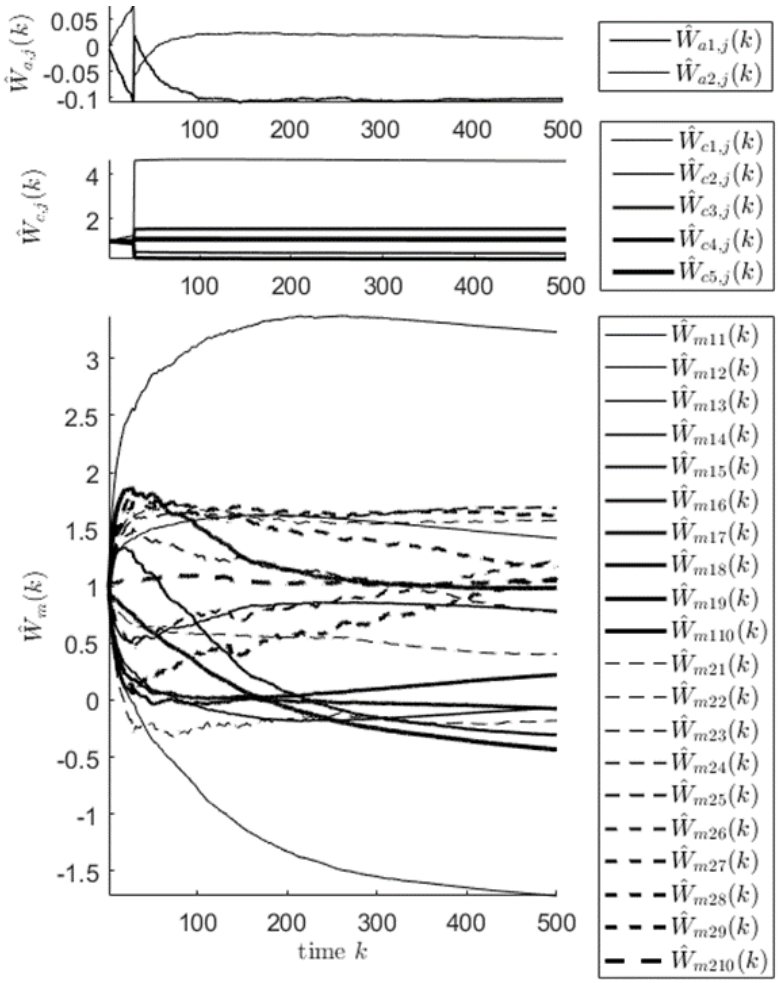

Figure 48. *The network weights $\widehat{W}_{a,j}(k)$, $\widehat{W}_{c,j}(k)$ and $\widehat{W}_m(k)$ with real-time data at each time step k*

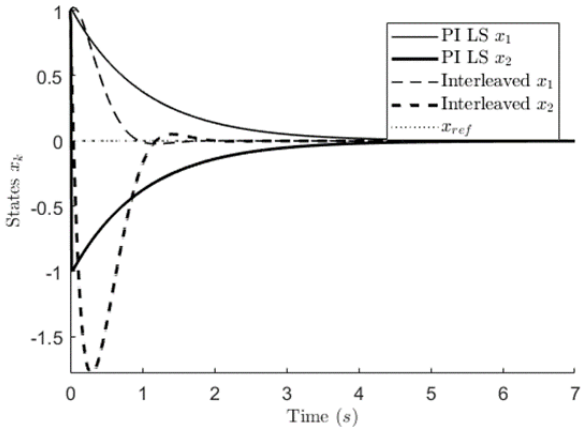

Figure 49. *The interleaved Q-learning policy that was learned using the real-time data is compared to the theoretical PI LS policy using the simulated model*

Figure 49 shows the learned interleaved Q-learning policy compared to the PI LS policy, when the policies are used in a simulated model.

4.4.2 On-policy real-time results

On-policy methods were used with the initial values and variables from Table 8.The simulator in Figure 10 with the real-time addition from Figure 15. Convergence limits were set as $\varepsilon_j = \varepsilon_i = 10^{-4}$ .The exploration noise was a uniform random noise between $[-5\,V, 5\,V]$ and control was limited between $[-7\,V, 7\,V]$. Figure 50, Figure 51 and Figure 52 show a case where learning is not successful (data 3 in Figure 46). Learning is considered successful here if it works for 200 s, while it might not be stable longer than that. Nothing is changed between runs.

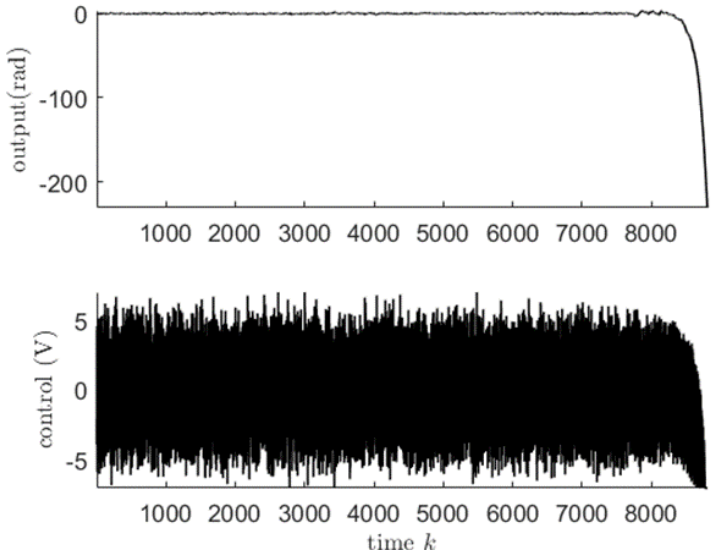

Figure 50. *Real-time control and output signals when the learning is unsuccessful with the PI RLS algorithm*

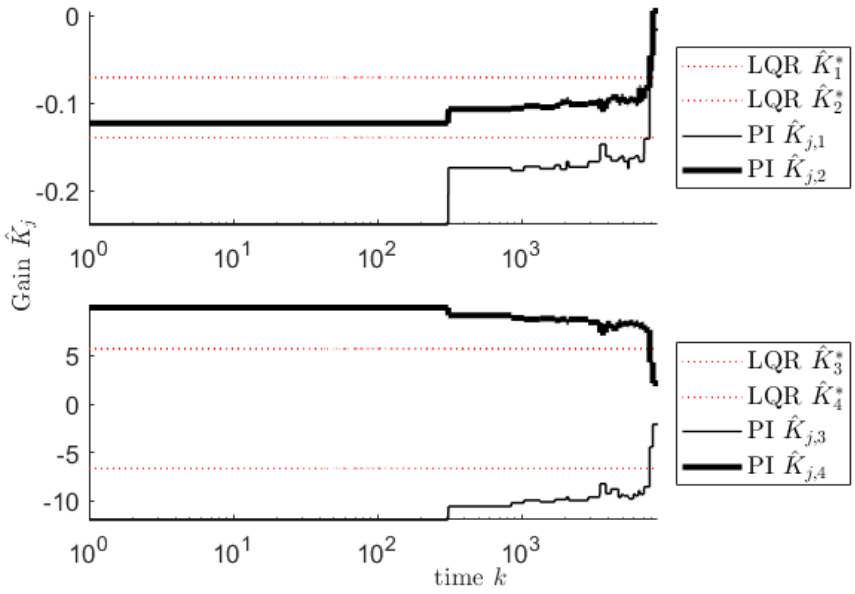

Figure 51. *Evolution of the PI RLS real-time gain $\hat{K}_j$ when the learning is unsuccessful (data 3)*

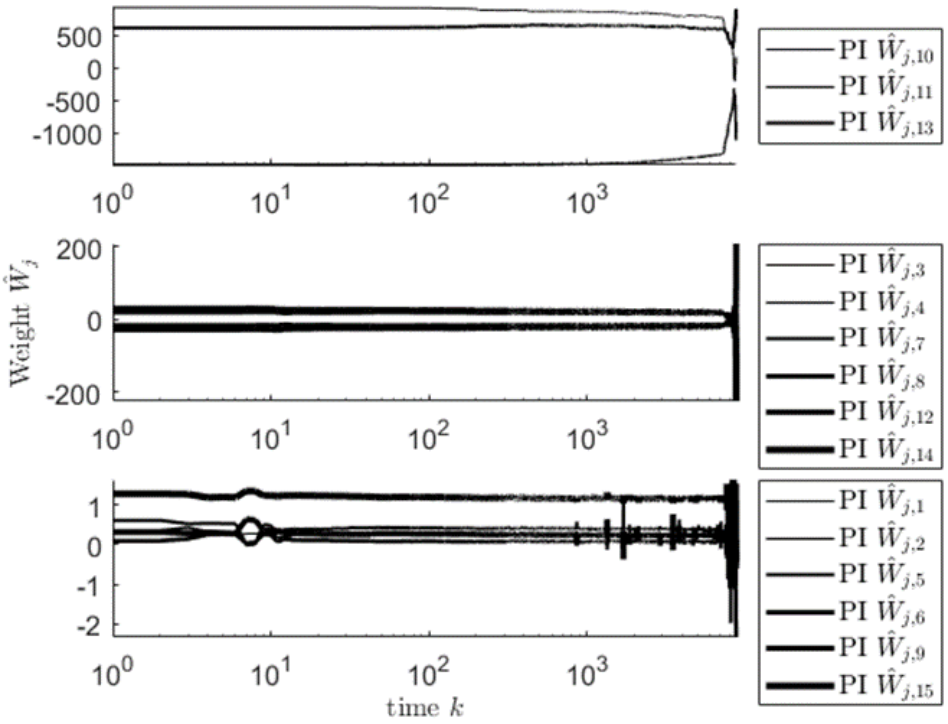

Figure 52. *Evolution of the PI RLS weights $\hat{W}_j$ when the learning is unsuccessful (data 3)*

Figure 53, Figure 54 and Figure 55 show a case where the learning is successful within the 200 seconds real-time run (data 1 and data 2 in Figure 46). The first 225 time steps of each trial were shown in Figure 46. The learned gains $\hat{K}_j$ of the successful trials are listed on Table 11.

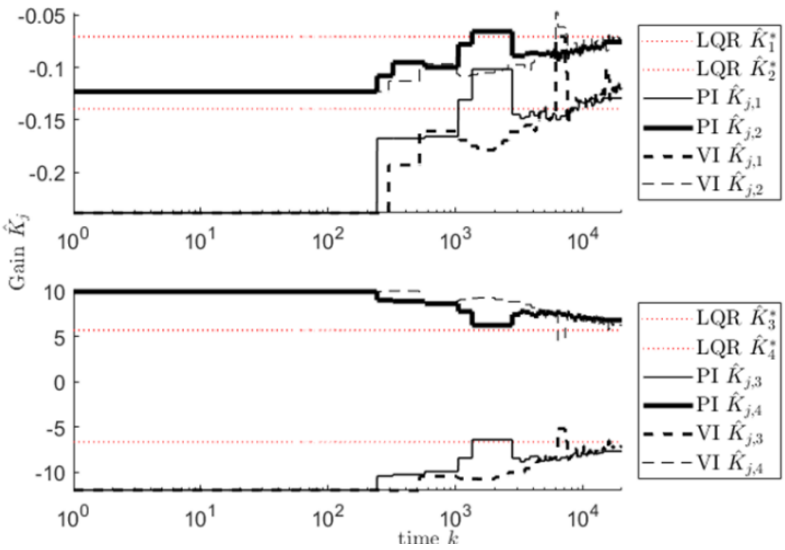

Figure 53. *Evolution of the PI and VI RLS gain $\hat{K}_j$ when the learning is successful*

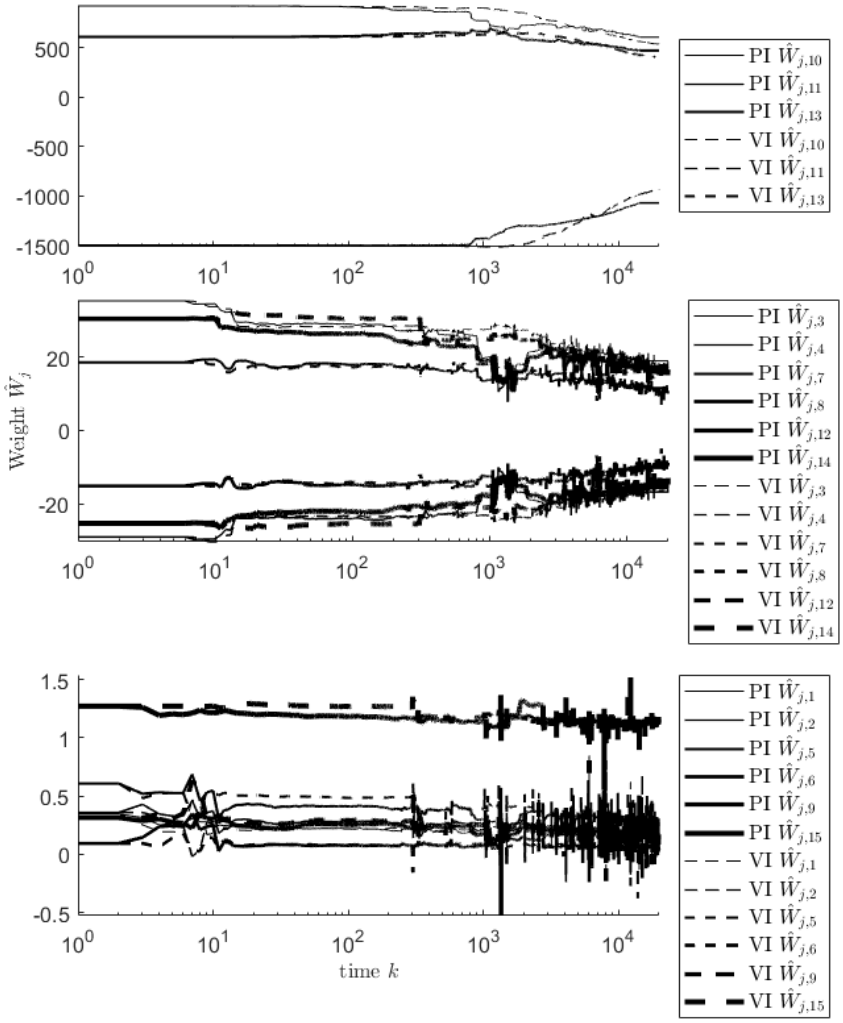

Figure 54. *Evolution of the PI and VI RLS weights $\hat{W}_j$ when the learning is successful*

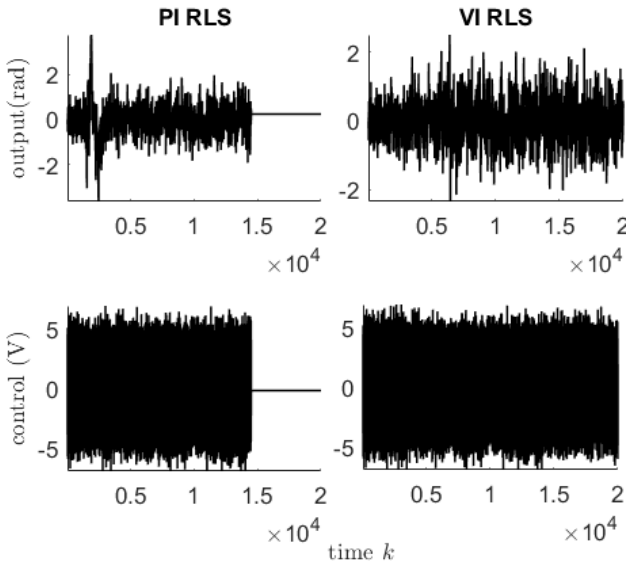

Figure 55. *Real-time control and output signals of the PI and VI RLS algorithm when the learning is successful*

Table 11. *On-policy real-time results if learning is successful within the time limit*

Algorithm	Output feedback gain $\hat{K}_j$
LQR ref.	$[-0.1395 \quad -0.0706 \quad -6.6362 \quad 5.7039]$
PI on-policy RLS	$[-0.1293 \quad -0.0755 \quad -7.6464 \quad 6.8042]$
VI on-policy RLS	$[-0.1198 \quad -0.0744 \quad -7.1140 \quad 6.2539]$

Figure 56, Figure 57 and Figure 58 show the results when on-policy PI and VI LS method is used and the learning is not successful within the 200 s running time. Convergence limits are changed to $\varepsilon_i = 10^{-1}$ and $\varepsilon_j = 10^{-5}$.

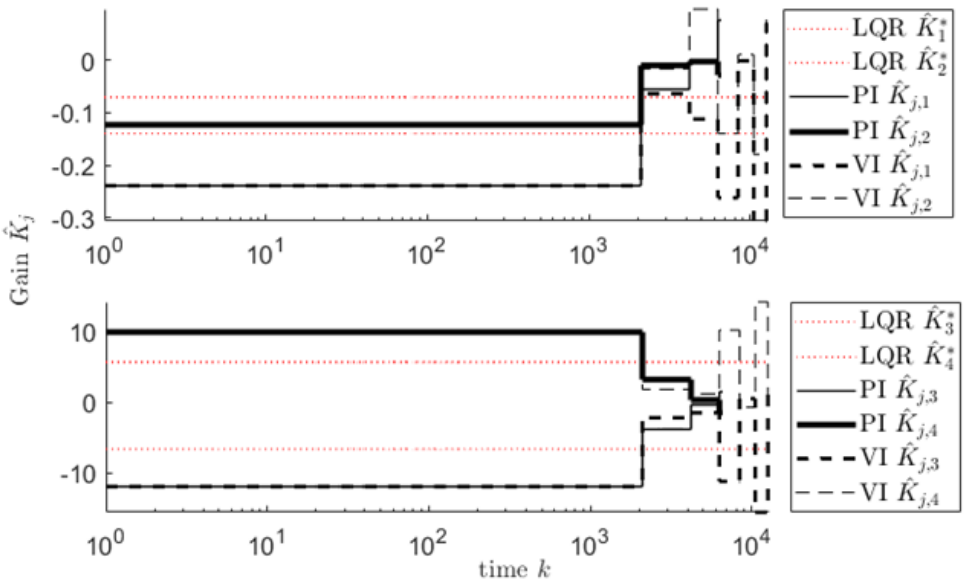

Figure 56. *Evolution of the on-policy PI and VI LS gain $\hat{K}_j$ in the real-time system*

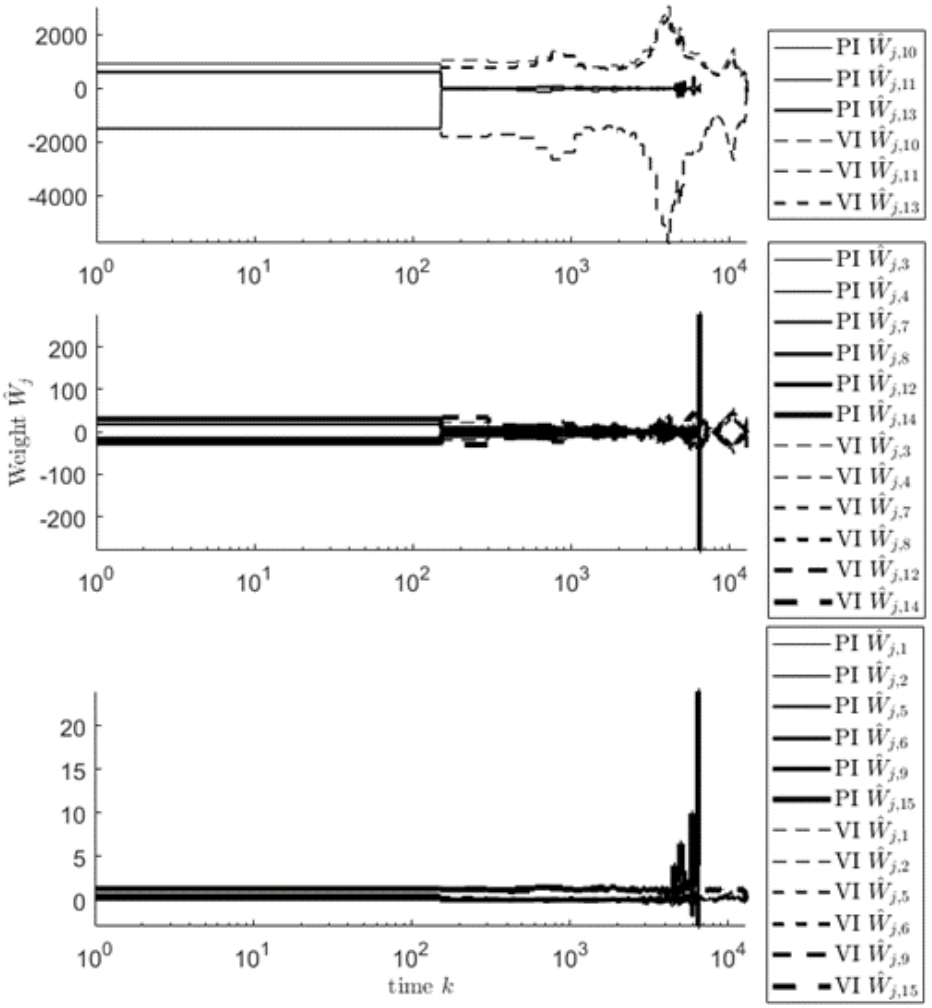

Figure 57. *Evolution of the on-policy PI and VI LS weight $\hat{W}_j$ in the real-time system*

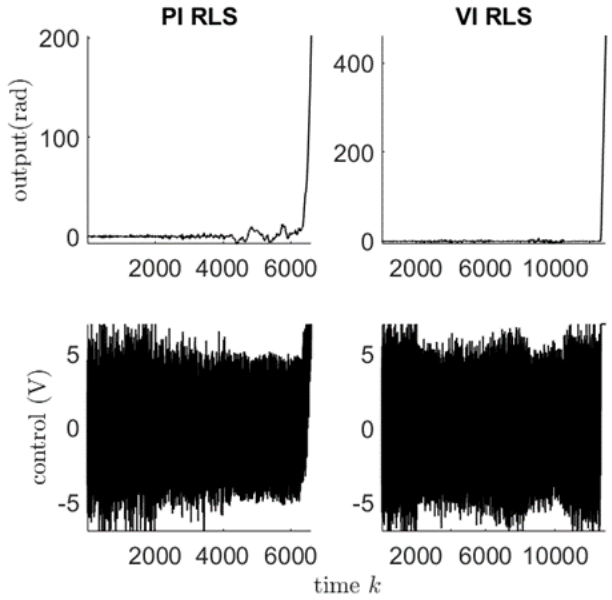

Figure 58. *Real-time output and control signals of the on-policy PI and VI LS*

Since the SGD algorithm only converges when using value iteration with full state measurements, the real-time control is applied with VI SGD full state algorithm only and the simulators in Figure 12 and Figure 16 are combined. Values from Table 2 are used and the learning rate is selected as $\alpha_{sgd} = 0.001$.

Table 12. *On-policy real-time results using SGD for full state estimation*

Algorithm	Full state gain $\widehat{K}_j$
LQR ref.	$[-0.4334 \quad -0.3957]$
VI on-policy SGD	$[-0.1571 \quad -0.007341]$

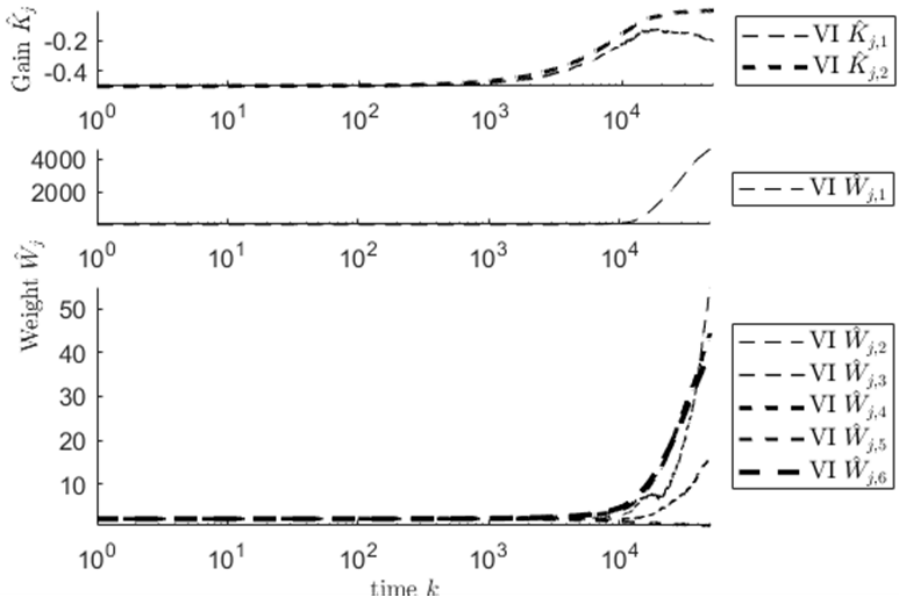

Figure 59. *Evolution of the gain $\widehat{K}_j$ and the weight $\widehat{W}_j$ with the full state VI SGD algorithm*

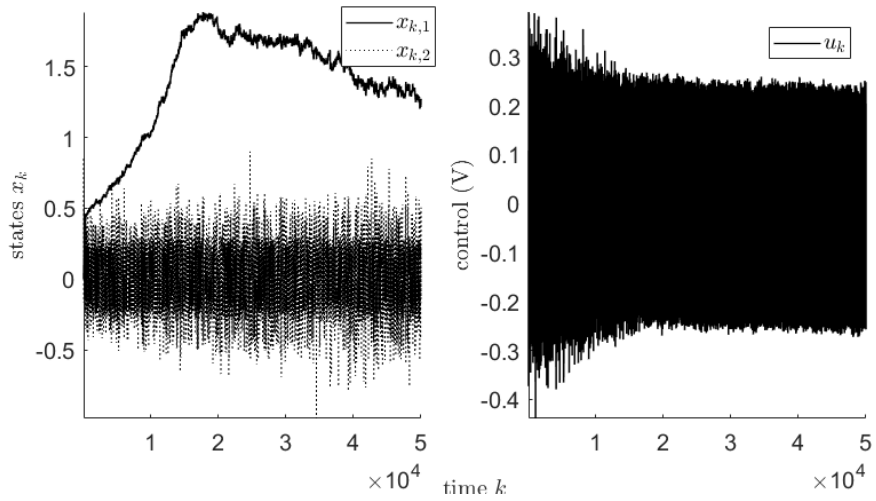

Figure 60. *Real-time output and control signals with the VI SGD algorithm*

Results from 500 s run are shown in Figure 59 and Figure 60. Table 12 shows the learned weights of the VI SGD algorithm.

4.4.3 Comparison between the real-time system and simulated results

Figure 61 shows the successful real-time system results next to the results with the modified simulator. The initial values were given in Chapter 4.4.2.

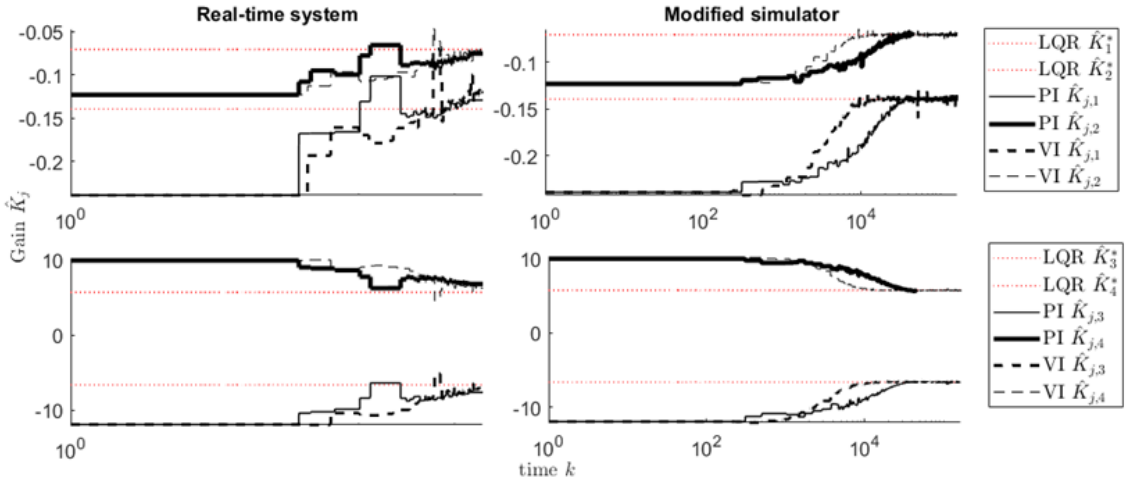

Figure 61. *The RLS PI and VI modified simulator results compared to the successful learning results from the real-time system*

It was seen that the behaviour of the real-time system was similar to the modified simulator, but the real-time system had unmodeled parts that affect the results.

4.5 Summary on Q-learning results in linear systems

Table 13 summarizes the results of all model-free LQR algorithms when a simulated model and output feedback measurements are used. These algorithms work as claimed in theory as they learn a policy near the model-based LQR solution. Only SGD algorithms did not find a solution when using the output feedback measurements.

Table 14 summarizes the real-time results of each model-free LQR algorithm. The results are the best trials out of all learning trials and the algorithms did not learn a proper solution each time when using the real-time system or they become unstable. On-policy learning was run for 500 s when using full-state SGD and for 200 s when using the other on-policy algorithms.

Interleaved Q-learning found a solution close to the optimal with the simulated system, but not with the real-time system. It was only used for full-state measurements.

Table 13. *The theoretical output feedback results with each algorithm*

Algorithm	Output feedback gain $\widehat{K}_j$
LQR ref.	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$
PI on-policy	
LS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$
SGD	Not found
RLS	$[-0.3654 \quad -0.1922 \quad -19.2503 \quad 15.5370]$
PI off-policy	
LS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$
LS2	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$
VI on-policy	
LS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$
SGD	Not found
RLS	$[-0.3654 \quad -0.1922 \quad -19.2498 \quad 15.5365]$
VI off-policy	
LS	$[-0.3654 \quad -0.1922 \quad -19.2496 \quad 15.5364]$
LS2	$[-0.3654 \quad -0.1922 \quad -19.2496 \quad 15.5364]$

Table 14. *The best real-time training trials of each algorithm*

Algorithm	Gain $\widehat{K}_j$
LQR ref.	$[-0.1395 \quad -0.0706 \quad -6.6362 \quad 5.7039]$ and $[-0.4334 \quad -0.3957]$ (full state)
PI on-policy	
LS	Not found
SGD	Not used
RLS	$[-0.1293 \quad -0.0755 \quad -7.6464 \quad 6.8042]$
PI off-policy	
LS	$[-0.0848 \quad -0.0509 \quad -4.8278 \quad 4.2489]$
LS2	$[-0.0445 \quad -0.0264 \quad -2.4660 \quad 2.2140]$
VI on-policy	
LS	Not found
SGD	$[-0.1571 \quad -0.007341]$ (full state)
RLS	$[-0.1198 \quad -0.0744 \quad -7.1140 \quad 6.2539]$
VI off-policy	
LS	$[-0.1112 \quad -0.0678 \quad -6.5225 \quad 5.6938]$
LS2	$[-0.1099 \quad -0.0662 \quad -6.2925 \quad 5.4742]$

The results in Table 14 prove that it is possible to adjust the exploration noise within the input control saturation limits so that the on-policy Q-learning algorithms converge near the optimal control with real-time data. However, the Q-learning algorithms are not stable on each run and therefore not reliable in real-time applications.

5. NONLINEAR SYSTEM Q-LEARNING RESULTS

The developed interleaved Q-learning algorithm was applied to the nonlinear the Quanser QUBE™-Servo 2 experiment with inverted pendulum attachment. First, the model from Appendix B was used to simulate data for comparison. Then, the simulated data was replaced by the real-time system data. The system is nonaffine, while the interleaved Q-learning algorithm was developed for affine systems in [19]. Near the upright position, the pendulum system was assumed affine. Therefore, only the balancing control was implemented model-free. The balancing control is calculated using the linearized system model in [1] and Appendix B.

5.1 Results with simulated data

The data was collected with a stabilizing behaviour policy $u_k = [3.2277 \quad -91.7893 \quad 3.9059 \quad -8.9131]x_k$ when the pendulum was balanced upwards using the simulated model. The exploration noise was selected as a sum of 31 sine waves of different frequencies and a uniform random noise within $[-0.2\,V \quad 0.2\,V]$. The structures for the model, the actor and the critic network were selected as 3-10-2, 3-4-1 and 3-10-1. The initial weights of the networks were chosen as $W_{a,0} = [0 \quad 20 \quad 0 \quad 0]^T$, $W_{c,0} = J_c$, where $J_c \in \mathbb{R}^{n_c x1}$ is a matrix of ones, and $W_{m,0} \in \mathbb{R}^{n_m x n_x}$ is a random matrix. Hidden layer weights were selected as in Program 6. The convergence limit was $\varepsilon_j = 5 \cdot 10^{-2}$. Learning rates were $\alpha_a = \alpha_c = 0.01$ and $\alpha_m = 0.4$. Figure 63 shows the evolution of the weights $\widehat{\mathrm{W}}_{\mathrm{a,j}}(\mathrm{k})$, $\widehat{\mathrm{W}}_{\mathrm{c,j}}(\mathrm{k})$ and $\widehat{\mathrm{W}}_{\mathrm{m}}(\mathrm{k})$.

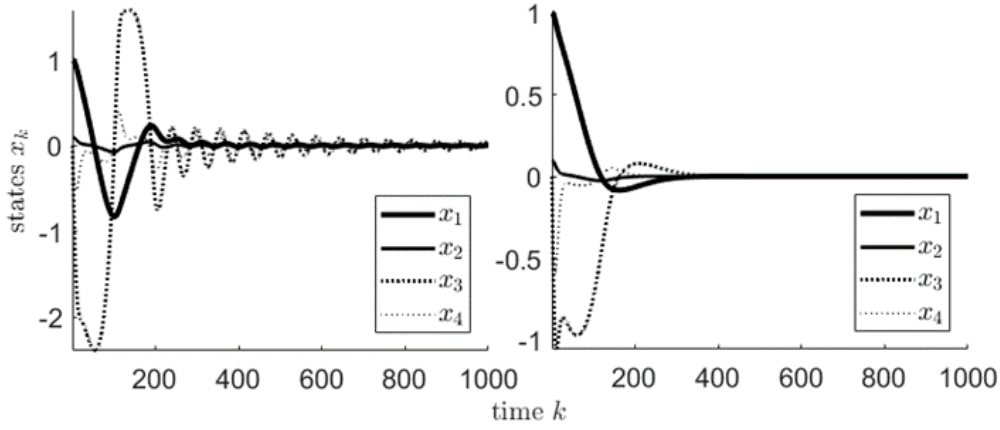

Figure 62. *The policy of the most successful learning trial used with the nonlinear system model (right) and linearized system model (left)*

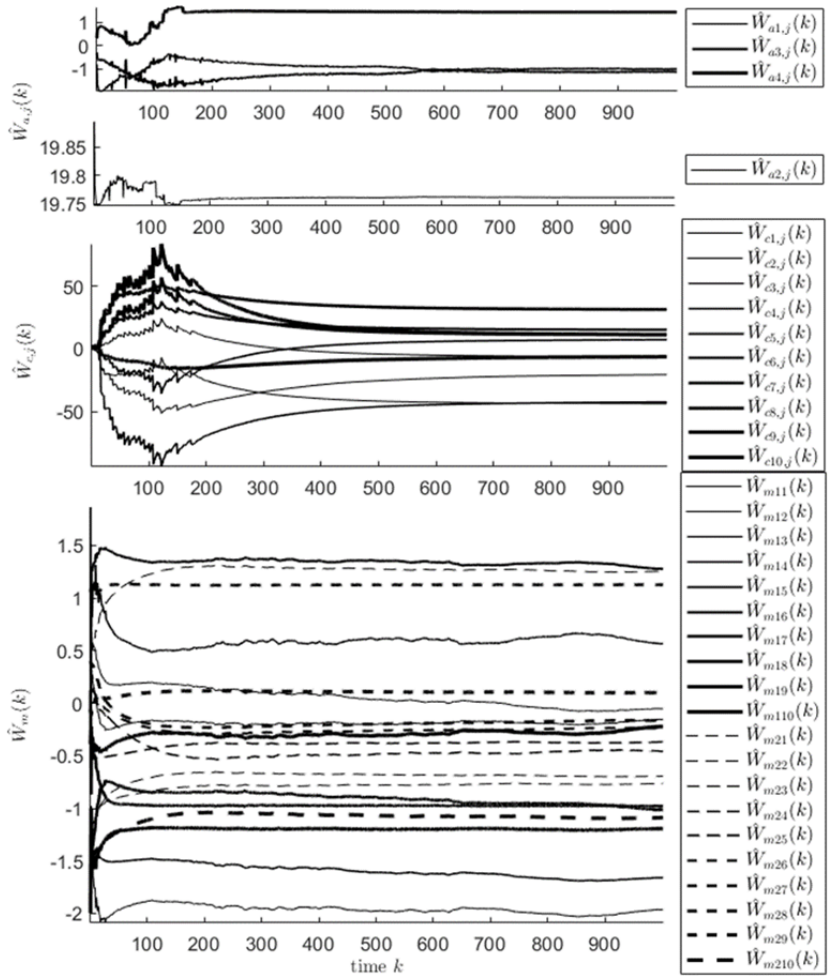

Figure 63. *Evolution of the interleaved Q-learning weights $\widehat{W}_{a,j}(k)$, $\widehat{W}_{c,j}(k)$ and $\widehat{W}_m(k)$*

Figure 62 shows the learned policy with the linearized and the nonlinear model from Appendix B with the initial state $x_0 = [0.1 \quad 1 \quad 0 \quad 0]^T$.The learned control policy only works in a limited region, close to the upright position. The final policy is $\hat{u}_{\infty,k} = -\widehat{W}_{a,\infty}^T \sigma(v_a^T x_k)$, where $\widehat{W}_{a,\infty} = [-0.7773 \quad 19.8255 \quad -0.9098 \quad 1.1146]^T$. The minus sign in the final policy is caused by the initialization of the actor weight $W_{a,0}$. Optimal policy was not found on each run.

5.2 Results with real-time data

The nonlinear real-time system replaced the simulated environment. Figure 64 shows the training data that is collected with the real-time system. The exploration noise and behaviour policy were selected the same as in Chapter 5.1. The angle measurements of each training episode are shown on Figure 64. These measurements were quantized and the control noise was increased to reduce the quantization error as was explained in Figure 36. The data in Figure 65 is collected with the increased uniform random noise

amplitude. Using larger exploration noise in real-time environment is challenging since balancing becomes hard. Therefore the data is shorter in Figure 65 than in Figure 64.

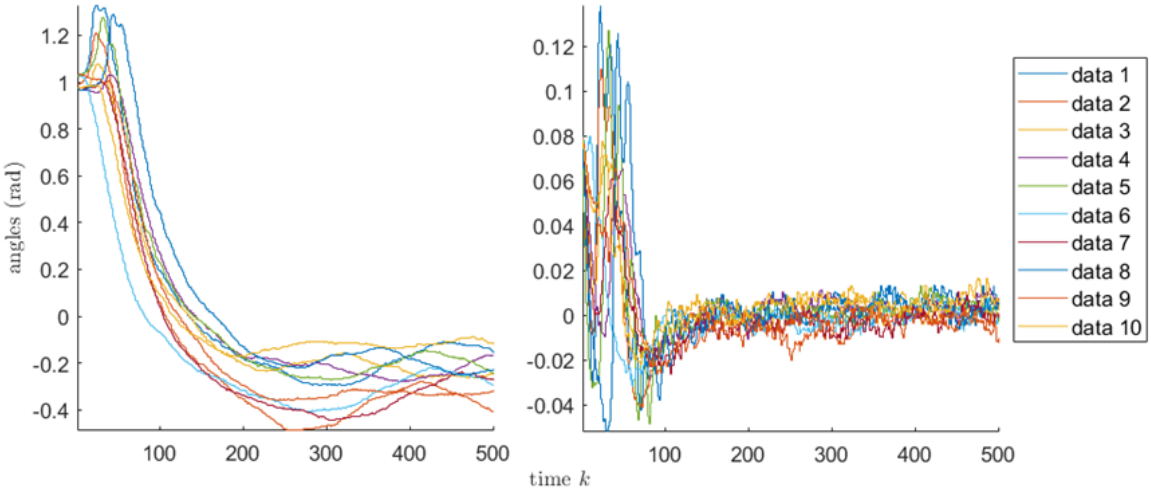

Figure 64. *The arm and pendulum angles of each training episode with the smaller noise*

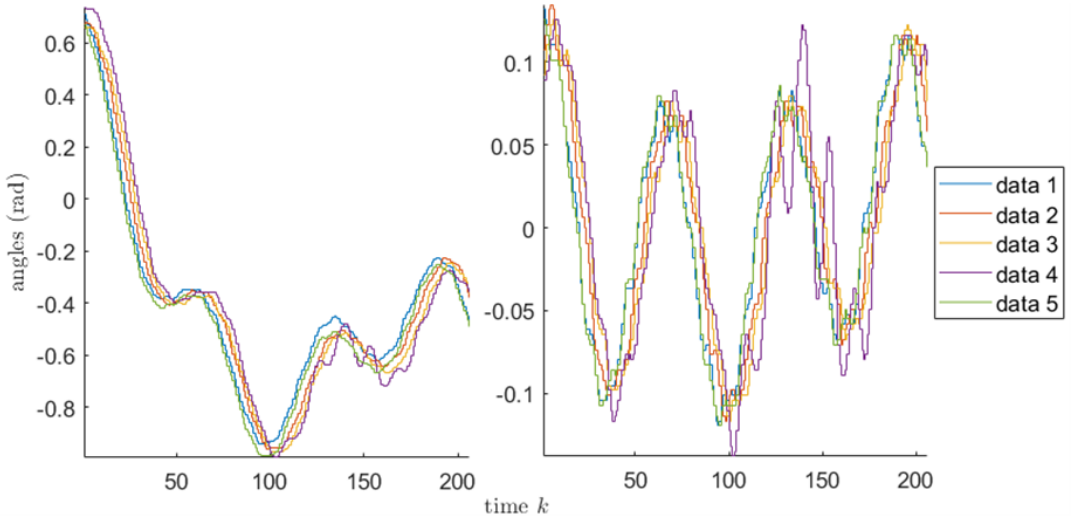

Figure 65. *The arm and pendulum angle of each training episode with the larger exploration noise*

The behaviour policy was calculated with a discretized version of the linearized model in Appendix B. Therefore, the learning region should have approximately $g(x_k) = B$, where B is the discrete-time input matrix. However, the estimated values of the model input dynamics $g(x_k)$ with (2.88) at time k are further away from the discretized linearized system model B matrix with the data from Figure 65 than with the data from Figure 64. The author was not able to find network structures and learning rates that would make the actor and critic weights converge with the real data in Figure 64 or in Figure 65.

CONCLUSIONS

This thesis studied model-free iterative Q-learning in real-time Quanser QUBE™-Servo 2 environment. The goal of the study was to determine if Q-learning can be applied to real-time systems when a system model is unknown by using a set of already existing Q-learning algorithms given in Figure 1. Linear system control was implemented using value and policy iteration algorithms with four different value update methods. Interleaved Q-learning was studied on both linear and nonlinear systems.

All of the theoretical algorithms were proven to work in a simulated environment with no disturbances as expected. They used either full-state measurements, output feedback measurements or both. However, gradient descent based methods (PI and VI SGD and Interleaved Q-learning) did not learn the optimal policy on each run. Instead, their success rates increase with a proper selection of the exploration noise, the batch size and the learning rates [32]. Also, the nonlinear system was a nonaffine system, but the selected algorithm was developed for affine systems and therefore the algorithm was only applied to a limited control region.

The theoretical linear system simulators were modified to resemble the real-time system. A constrained control input, quantized measurements and measurement disturbances were added to the system. With a small noise and a quantization present in the modified simulator, the on-policy algorithms converged near the optimal control value with a proper noise selection. When the real-time system was used, on-policy PI and VI RLS methods converged to the correct solution on few runs, but several times the learning was unreliable and the system became unstable.

Off-policy value iteration methods found a near optimal control solution with all of the tested measurement datasets from the real-time system while off-policy policy iteration methods did not find a near optima solution each time with the same data. Interleaved Q-learning found a policy when using the real linear system data, but the policy was not optimal. The author did not manage to find an exploration noise and learning rates that would make the interleaved Q-learning algorithm converge with the data from the real-time inverted pendulum.

The results proved that model-free feedback control with Q-learning could be used in linear real-time environments in the future, but more research need to be conducted on how to make on-policy learning more reliable. Certain issues such as quantized measurements can be removed with the selection of the exploration noise, but in long term

applications large changes in the control voltage can harm the system. Off-policy PI and VI methods are not adaptive, but based on the results they can learn the optimal control with a smaller exploration noise and with less data than the on-policy methods. Q-learning is not widely studied in nonlinear applications yet and the algorithms lack proof of stability and unbiasedness.

One of the main problems of the ADP algorithms is that there are no sophisticated methods to find a proper exploration noise nor an initial stabilizing policy for policy iteration and interleaved Q-learning. With poor selection of the exploration noise, the system might converge to a non-optimal policy as in Figure 42. Without any knowledge of the system, one does not know if the learned control is the optimal. Similar problems are caused if the initial control policy is not stabilizing for policy iteration.

New algorithms must be developed. First, more Q-learning applications with nonaffine systems must be developed. Then, new algorithms must consider the disturbances, the quantized measurements and the quantized and saturated control inputs. Some literature (see [5][18][45]) has discussed these topics, but barely any information is found on measurement noise or other disturbances. These limitations of the real-time system with a lack of methods to select the exploration noise and initial control policy still remain the largest obstacles for Q-learning applications in real-time feedback control.

REFERENCES

[1] J. Apkarian et al, Student Workbook: QUBE-Servo 2 Experiment for MATLAB®/Simulink® Users, Available: https://www.quanser.com/courseware-resources/ (01.03.2019)

[2] T. Dierks and S. Jagannathan, Online Optimal Control of Affine Nonlinear Discrete-Time Systems With Unknown Internal Dynamics by Using Time-Based Policy Update, IEEE Transactions on Neural Networks and Learning Systems, vol. 23, no. 7, 2012, pp. 1118-1129.

[3] T. Dierks, B. T. Thumati and S. Jagannathan, Optimal control of unknown affine nonlinear discrete-time systems using offline-trained neural networks with proof of convergence, Neural Networks, vol. 22, no.5, 2009, pp. 851-860.

[4] D. Ding et al, Neural-Network-Based Output-Feedback Control Under Round-Robin Scheduling Protocols, IEEE Transactions on Cybernetics, vol. 49, no.6, 2019, pp. 2372-2384.

[5] J. Duan, H. Xu and W. Liu, Q-Learning-Based Damping Control of Wide-Area Power Systems Under Cyber Uncertainties, IEEE Transactions on Smart Grid, vol. 9, no. 6, 2018, pp. 6408-6418.

[6] G. F. Franklin, J. D. Powell and M. L. Workman, Digital Control of Dynamic Systems, 3rd edition, 1998, Addison-Wesley, 742 p.

[7] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning. Cambridge, MIT Press, 2017, pp. 149-150, 274-302.

[8] S. ten Hagen and B. Kröse, Neural Q-learning, Neural Computing&Applications, vol. 12, no. 2, 2003, pp. 81-88.

[9] P. C. Hansen, V. Pereyra and G. Scherer, Least Squares Data Fitting with Applications, 2013, John Hopkins University Press, 305 p.

[10] L. Hager et al, Adaptive Neural network control of a helicopter system with optimal observer and actor-critic design, Neurocomputing, vol. 302, 2018, pp. 75-90.

[11] P. He and S. Jagannathan, Reinforcement Learning Neural-Network-Based Controller for Nonlinear Discrete-Time Systems With Input Constraints, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 37,no. 2, 2007, pp. 425-436.

[12] A. Heydari, Revisiting Approximate Dynamic Programming and its Convergence, IEEE Transactions on Cybernetics, vol. 44, no. 12, 2014, pp. 2733-2743.

[13] Y. Jiang et al, Data-Driven Flotation Industrial Process Operational Optimal Control Based on Reinforcement Learning, IEEE Transactions on Industrial Informatics, vol. 14, no.5, 2018, pp. 1974-1989.

[14] B. Kiumarsi et al, Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics, Automatica, vol. 50, no. 4, 2014, pp. 1167-1175.

[15] F. L. Lewis and K. G. Vamvoudakis, Reinforcement Learning for Partially Observable Dynamic Processes: Adaptive Dynamic Programming Using Measured Output Data, IEEE Transactions on Systems, Man, and Cybernetics, vol. 41, no. 1, 2011, pp. 14-25.

[16] F. L. Lewis and D. Vrabie, Reinforcement learning and adaptive dynamic programming for feedback control, IEEE Circuits and Systems Magazine, vol. 9, no. 3, 2009, pp. 32-50.

[17] F. L. Lewis, D. Vrabie, K. G. Vamvoudakis, Reinforcement learning and Feedback Control, IEEE Control Systems, vol. 32, no. 6, 2012, pp. 76-105.

[18] Y. Liang et al, A neural network-based approach for solving quantized discrete-time H∞ optimal control with input constraint over finite-horizon, Neurocomputing, vol. 333, 2019, pp. 248-260.

[19] J. Li et al, Off-Policy Interleaved Q-Learning: Optimal Control for Affine Nonlinear Discrete-Time Systems, IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 5, 2019, pp. 1-13.

[20] J. Li et al, Off-Policy Q-Learning: Set-Point Design for Optimizing Dual-Rate Rougher Flotation Operational Processes, IEEE Transactions on Industrial Electronics, vol. 65, no. 5, 2018, pp. 4092-4102.

[21] J. Li et al, Off-Policy Reinforcement Learning: Optimal Operational Control for Two-Time-Scale Industrial Processes, IEEE Transactions on Cybernetics, vol. 47, no. 12, 2017, pp. 4547-4558.

[22] Y. Li et al, Data-driven approximate Q-learning stabilization with optimality error bound analysis, Automatica, vol. 103, 2019, pp. 435-442.

[23] H. Lin, Q. Wei, D. Liu, Online identifier-actor-critic algorithm for optimal control of nonlinear systems, Optimal Control Applications and Methods, vol. 38, no. 3, 2017, pp. 317-335.

[24] D. Liu et al, Neural-Network-Based Optimal Control for a Class of Unknown Discrete-Time Nonlinear Systems Using Globalized Dual Heuristic Programming, IEEE Transactions on Automation Science and Engineering, vol. 9, no. 3, 2012, pp. 628-634.

[25] D. Liu and Q. Wei, Policy Iteration Adaptive Dynamic Programming Algorithm for Discrete-Time Nonlinear Systems, IEEE Transactions on Neural Networks and Learning Systems, vol. 25, no. 3, 2014, pp. 621-634.

[26] B. Luo et al, Model-Free Optimal Tracking Control via Critic-Only Q-Learning, IEEE Transactions on Neural Networks and Learning Systems, 2016, vol. 27, no. 10, pp. 2134-2144.

[27] B. Luo et al, Policy Gradient Adaptive Dynamic Programming for Data-Based Optimal Control, IEEE Transactions on Cybernetics, vol. 47, no. 10, 2017, pp. 3341-3354.

[28] M. N. Nounou, H. N. Nounou and M. Mansouri, Model-based and model-free filtering of genomic data, Network Modeling and Analysis in Health Informatics and Bioinformatics, vol. 2, no. 3, 2013, pp. 109-121.

[29] M. Park, Y. Kim and J. Lee, Swing-up and LQR stabilization of a rotary inverted pendulum, Artificial Life and Robotics, vol. 16, no. 1, 2011, pp. 94-97.

[30] Quanser Inc., User Manual: QUBE-Servo 2 Experiment Set Up and Configuration

[31] M. Radac and R. Precup, Data-driven model-free slip control of anti-lock braking systems using reinforcement Q-learning, Neurocomputing, vol. 275, 2018, pp. 317-329.

[32] M. Riedmiller, Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method, 16th European Conference on Machine Learning (ECML), Oct 2005, pp. 317-328.

[33] S. A. A. Rizvi and Z. Lin, An iterative Q-learning scheme for the global stabilization of discrete-time linear systems subject to actuator saturation, International Journal of Robust and Nonlinear Control, vol. 29, no. 9, 2019, pp. 2660-2672.

[34] S. A. A. Rizvi and Z. Lin, Model-Free Global Stabilization of Discrete-Time Linear Systems with Saturating Actuators Using Reinforcement Learning, 2018 IEEE Conference on Decision and Control (CDC), Miami Beach, FL, 2018, pp. 5276-5281.

[35] S. A. A. Rizvi and Z. Lin, Output feedback optimal tracking control using reinforcement Q-learning, 2018 Annual American Control Conference (ACC), Jun 2018, pp. 3423-3428.

[36] S. A. A. Rizvi, Z. Lin, Output Feedback Reinforcement Q-Learning Control for the Discrete-Time Linear Quadratic Regulator, IEEE 56th Annual Conference on Decision and Control (CDC), Dec 2017, pp. 1311-1316.

[37] R. Song et al, Multiple Actor-Critic Structures for Continuous-Time Optimal Control Using Input-Output Data, IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 4, 2015, pp. 851-865.

[38] R. Song et al, Off-Policy Actor-Critic Structure for Optimal Control of Unknown Systems With Disturbances, IEEE Transactions on Cybernetics, vol. 46, no.5, 2016, pp. 1041-1050.

[39] R. T. Sukhavasi and B. Hassibi, The Kalman-Like Particle Filter: Optimal Estimation With Quantized Innovations/Measurements, IEEE Transactions on Signal Processing, vol. 61, no. 1, 2013, pp. 131-136.

[40] D. Wang et al, Optimal control of unknown nonaffine nonlinear discrete-time systems based on adaptive dynamic programming, Automatica, vol. 48, no. 8, 2012, pp. 1825-1832.

[41] Q. Wei and D. Liu, A Novel Iterative θ -Adaptive Dynamic Programming for Discrete-Time Nonlinear Systems, IEEE Transactions on Automation Science and Engineering, vol. 11, no. 4, 2014 pp. 1176-1190.

[42] Q. Wei, D. Liu, A novel policy iteration based deterministic Q-learning for discrete-time nonlinear systems, Science China Information Sciences, vol. 58, no. 12, 2015, pp. 1-15.

[43] Q. Wei, D. Liu and H. Lin, Value Iteration Adaptive Dynamic Programming for Optimal Control of Discrete-Time Nonlinear Systems, IEEE Transactions on Cybernetics, vol. 46, no. 3, 2016, pp. 840-853.

[44] Q. Wei, R. Song and Q. Sun, Nonlinear neuro-optimal tracking control via stable iterative Q-learning algorithm, Neurocomputing, vol. 168, 2015, pp. 520-528.

[45] Q. Zhao, H. Xu and S. Jagannathan, Optimal control of uncertain quantized linear discrete-time systems, International Journal of Adaptive Control and Signal Processing, vol. 29, no. 3, 2015, pp. 325-345.

APPENDIX A: QUANSER QUBE™-SERVO 2: INERTIA DISK ATTACHMENT

Quanser QUBE™-Servo 2 rotary servo experiment documentation [1] gives motion models to the DC motor with load. The system parameters are given in table A1.

Table A1. DC motor parameters, modified from documentation [1]

Parameter	Value
Terminal resistance R_m	$8.4\ \Omega$
Torque constant k_t	$0.042\ Nm/A$
Motor back-emf constant k_m	$0.042\ V/(rad/s)$
Rotor inertia J_m	$4.0 \cdot 10^{-6}\ kgm^2$
Rotor inductance L_m	$1.16\ mH$
Load hub mass m_h	$0.0106\ kg$
Load hub radius r_h	$0.0111\ m$
Load hub inertia J_h	$0.6 \cdot 10^{-6}\ kgm^2$

Table A2. Load disk parameters, modified from documentation [1]

Parameter	Value
Mass of disk load m_d	$0.053\ kg$
Radius of disk load r_d	$0.0248\ m$

A two-state system with position and angular velocity as the states gives

$$x = [\theta \quad \dot{\theta}]^T \tag{A.1}$$

where θ is the position in radians and $\dot{\theta}(t)$ is the angular velocity $\omega_m(t) = \dot{\theta}(t)$. Only the position is measured so that output y is $y = [1 \quad 0]x$.

Current is given in documentation [1] as

$$i_m(t) = \big(v_m(t) - k_m\omega_m(t)\big)/R_m\,, \tag{A.1}$$

where v_m is the input voltage v_m and the control input is $u(t) = v_m(t)$. Motor shaft equation is given as

$$J_{eq}\dot{\omega}_m(t) = k_t i_m(t) \tag{A.2}$$

$$J_{eq} = J_m + J_h + \tfrac{1}{2}m_d r_d^2 \tag{A.3}$$

where J_m is rotor moment of inertia, J_h is load hub inertia, $m_d$is inertia disk mass and r_d is inertia disk radius.

Using equation $(A.2)$ to solve $\dot{\omega}_m(t)$ and replacing $i_m(t)$ with equation $(A.1)$ yields

$$\dot{\omega}_m(t) = \ddot{\theta}(t) = -(k_t k_m)/R_m J_{eq}\,\dot{\theta}(t) + k_t/R_m J_{eq}\,\dot{\theta}u(t) \qquad (A.4)$$

Using $(A.1)$ and $(A.4)$ the state derivative $\dot{x}$ becomes

$$\dot{x} = \begin{bmatrix}\dot{\theta}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & -\frac{k_t k_m}{R_m J_{eq}}\end{bmatrix} x + \begin{bmatrix}0\\ \frac{k_t}{R_m J_{eq}}\end{bmatrix} u \qquad (A.5)$$

This is the continuous-time model that can be used to generate data for simulation purposes. The eigenvalues of $A = \begin{bmatrix}0 & 1\\ 0 & -\frac{k_t k_m}{R_m J_{eq}}\end{bmatrix}$ are 0 and $-\frac{k_t k_m}{R_m J_{eq}} = -10.0485.$

APPENDIX B: QUANSER QUBE™-SERVO 2: INVERTED PENDULUM ATTACHMENT

Quanser QUBE™-Servo 2 rotary servo experiment DC motor parameters were given in Table A1. The parameters of the pendulum attachment are listed in Table B1.

Table B1. Rotary arm and pendulum link parameters, modified from documentation [1]

Parameter	Value
Rotary arm mass m_r	$0.095\ kg$
Rotary arm length l_r	$0.085\ m$
Rotary arm moment of inertia J_r	$5.720 \cdot 10^{-5}\ kgm^2$
Rotary arm equivalent viscous damping coefficient D_r	$0.0015\ Nms/rad$
Pendulum link mass m_p	$0.024\ kg$
Pendulum link total length l_p	$0.129\ m$
Pendulum link moment of inertia J_p	$3.33 \cdot 10^{-5}\ kgm^2$
Pendulum link equivalent viscous damping coefficient D_p	$0.0005\ Nms/rad$
Gravitational constant g	$9{,}81\ m/s^2$

From the Quanser manual [1] the system model is defined as

$$\ddot{\theta} = \frac{(c\cos\beta)\ddot{\beta} - (2a\sin\beta\cos\beta)\dot{\theta}\dot{\beta} - (c\sin\beta)\,\dot{\beta}^2 + \tau - D_r\dot{\theta}}{b + a - a(\cos\beta)^2 + J_r} \qquad (B.1)$$

and

$$\ddot{\beta} = \frac{-(c\cos\beta)\ddot{\theta} + (a\cos\beta\sin\beta)\dot{\theta}^2 - d\sin\beta - D_p\dot{\beta}}{J_p + a} \qquad (B.2)$$

where the constants are

$$a = \frac{1}{4}m_p{l_p}^2,\ b = m_p{l_r}^2,\ c = \frac{1}{2}m_p l_p l_r,\ d = \frac{1}{2}m_p l_p g \qquad (B.3)$$

and the torque τ at the base of the rotary arm is

$$\tau = k_t\left(V_m - k_m\dot{\theta}\right)/R_m \qquad (B.4)$$

where V_m is the control voltage.

Equations $(B.1)$ and $(B.2)$ can be marked as

$$\ddot{\theta} = \left(H\ddot{\beta} + Y\right)/X \qquad (B.5)$$

$$\ddot{\beta} = (-H\ddot{\theta} + Z)/G \qquad (B.6)$$

where

$$\begin{cases} Y = -(2a\sin\beta\cos\beta)\dot{\theta}\dot{\beta} - (c\sin\beta)\,\dot{\beta}^2 + \tau - D_r\dot{\theta} \\ X = b + a - a(\cos\beta)^2 + J_r \\ H = (c\cos\beta) \\ Z = (a\cos\beta\sin\beta)\dot{\theta}^2 - d\sin\beta - D_p\dot{\beta} \\ G = J_p + a \end{cases} \quad (B.7)$$

Inserting equation $(B.6)$ into $(B.5)$ yields

$$\ddot{\theta} = \frac{H(-H\ddot{\theta}+Z)/G+Y}{X} \quad (B.8)$$

The angular acceleration $\ddot{\theta}$ is solved from equation $(B.15)$ as

$$\ddot{\theta} = (HZ/G + Y)/(X + H^2/G) \quad (B.9)$$

States can be chosen as

$$x = [\theta \quad \beta \quad \dot{\theta} \quad \dot{\beta}]^T = [x_1 \quad x_2 \quad x_3 \quad x_4]^T \quad (B.10)$$

The first two states, the body angle and pendulum angle are measurable, but velocities can be estimated with different filters. If the system model can is linearized at the top position with $\beta = 0$, The linearized continuous-time model is derived in [1] as

$$\dot{x} = \frac{1}{J_t}\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & cd & -\left(D_r + {}^{k_m}/_{R_m}\right)G & -cD_p \\ 0 & d(J_r+b) & -c\left(D_r + {}^{k_m}/_{R_m}\right) & -D_p(J_r+b) \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \frac{k_m}{J_t R_m}\begin{bmatrix} 0 \\ 0 \\ G \\ c \end{bmatrix} u \quad (B.11)$$

where

$$J_t = J_r J_p + aJ_r + bJ_p + c^2 \quad (B.12)$$

To balance the pendulum in an upright position, swing up control is used when the pendulum is not near the equilibrium position as in [1]. Otherwise, the balancing control is used. Swing up control is derived as energy control where the control policy is

$$u = \mathrm{sat}_{u_{max}}\left(K_p(E_r - E)\mathrm{sign}\left(\dot{\beta}\cos\beta\right)\right) \quad (B.13)$$

and the energy is

$$E = \frac{1}{2}J_p\dot{\beta}^2 + \frac{1}{2}m_p g l_p(1-\cos\beta) \quad (B.14)$$

E_r is the reference energy and setting it equal to the potential energy E_p will swing the pendulum into an upright position and K_p is a tuneable control gain for the pendulum attachment.